University of the Philippines Manila

College of Arts and Sciences

Department of Physical Sciences and Mathematics

Toxicheck: In-Silico Nano-QSAR Toxicity Classification using Hybrid Machine Learning Algorithms

A special problem in partial fulfillment

of the requirements for the degree of

Bachelor of Science in Computer Science

Submitted by:

John Derick E. Barcellano

June 2023

Permission is given for the following people to have access to this SP:

Available to the general public	Yes
Available only after consultation with author/SP adviser	No
Available only to those bound by confidentiality agreement	No

ACCEPTANCE SHEET

The Special Problem entitled "Toxicheck: In-Silico Nano-QSAR Toxicity Classification using Hybrid Machine Learning Algorithms" prepared and submitted by John Derick E. Barcellano in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science has been examined and is recommended for acceptance.

Perlita E. Gasmen, M.Sc.
Adviser

EXAMINERS:

	Approved	**Disapproved**
1. Avegail D. Carpio, M.Sc.		
2. Richard Bryann Chua, Ph.D (*cand.*)		
3. Ma. Sheila A. Magboo, Ph.D. (*cand.*)		
4. Vincent Peter C. Magboo, M.D., M.Sc.		
5. Marbert John C. Marasigan, M.Sc. (*cand.*)		
6. Geoffrey A. Solano, Ph.D.		

Accepted and approved as partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science.

Vio Jianu C. Mojica, M.Sc.
Unit Head
Mathematical and Computing Sciences Unit
Department of Physical Sciences
and Mathematics

Marie Josephine M. De Luna, Ph.D.
Chair
Department of Physical Sciences
and Mathematics

Maria Constancia O. Carrillo, Ph.D.
Dean
College of Arts and Sciences

Abstract

The health hazards and risks of nanoparticles (NPs) and engineered nanomaterials (ENMs) are linked to their physicochemical features. Due to their minute structure, they can cause intracellular and genetic damage, and harm the environment by forming toxic mixtures with other compounds. Thus, it is essential to assess them first before they are mass-produced for public use. Traditionally, nanomaterial toxicity involves in-vivo and in-vitro approaches, but in recent years, machine learning (ML) algorithms have also emerged as predictive tools through in-silico means. This approach provides a faster, cheaper, and safer way to assess the toxicological profile of a nanomaterial. This study aims to investigate the applicability and efficiency of using hybrid algorithms in nanomaterial toxicity classification. They are formed by combining Genetic Algorithm (GA) with different base classifiers, namely Logistic Regression (LR), Artificial Neural Network (ANN), and Random Forest (RF). Generally, the hybrid algorithm-based models perform better than their base classifier counterparts, with an increase in scores of up to 19%. Using MCC as the main metric, results show that GA-RF with SMOTE is the best-performing model with an MCC score of 0.34. Building upon this model, this study developed a web application that lets the user input information about a nanomaterial and the cell-based assay that will be exposed for a certain amount of time. It predicts the cell viability of the assay to produce a toxicity classification for the nanomaterial.

Keywords: Nanomaterial toxicity, hybrid algorithm, genetic algorithm, cell viability, machine learning

Contents

Acceptance Sheet i

Abstract ii

List of Figures vi

List of Tables vii

I. Introduction 1

A. Background of the Study . . . 1

B. Statement of the Problem . . . 2

C. Objectives of the Study . . . 3

D. Significance of the Project . . . 4

E. Scope and Limitations . . . 4

F. Assumptions . . . 5

II. Review of Related Literature 6

A. Nanomaterials . . . 6

B. Testing Approaches . . . 8

C. Machine Learning . . . 9

III. Theoretical Framework 12

A. Nanomaterial Toxicity . . . 12

B. In-Silico Toxicological Testing . . . 12

B.1. Quantitative Structure-Activity Relationship . . . 12

C. Synthetic Minority Oversampling Technique (SMOTE) . . . 13

D. Feature Scaling . . . 13

E. Hyperparameter Tuning . . . 14

F. Machine Learning . . . 14

F.1. Feature Selection 15

F.2. Classifiers 16

G. Performance Metrics 17

G.1. Accuracy 17

G.2. Precision 17

G.3. Recall 17

G.4. F1 Score 18

G.5. Receiver Operating Characteristic (ROC) 18

G.6. Area Under the Curve (AUC) 18

G.7. Matthew's Correlation Coefficient (MCC) 19

IV. Design and Implementation **20**

A. Dataset 20

B. General Workflow 21

B.1. Data Preprocessing 21

B.2. Feature Selection using Genetic Algorithm 22

B.3. Model Implementation 22

B.4. Model Evaluation and Application 23

C. Use-Case Diagram 23

D. Context-Free Diagram 24

E. System Architecture 24

F. Technical Architecture 25

V. Results **26**

A. Data Preprocessing 26

B. Feature Selection using Genetic Algorithm 28

C. Data Splitting and Class Balancing using SMOTE 29

D. Feature Scaling 30

E. Hyperparameter Tuning using Grid Search 30
F. Model Evaluation 31
F.1. Model Performance with SMOTE and Genetic Algorithm . . . 31
F.2. Model Performance with Feature Scaling 33
F.3. Model Performance with All Applied 33
G. Web-Based Application 34
G.1. Landing Page 34
G.2. Input Form Page 35
G.3. Results View 36

VI. Discussions **37**

VII. Conclusions **39**

VIII. Recommendations **40**

IX. Bibliography **41**

X. Appendix **44**
A. Source Code 44

XI. Acknowledgment **64**

List of Figures

1 Schematic representation of the QSAR modeling workflow. 13

2 Genetic algorithm workflow. 15

3 Accuracy rate formula. 17

4 Precision rate formula. 17

5 Recall rate formula. 18

6 F1 score formula. 18

7 Matthew's correlation coefficient score formula. 19

8 General workflow of the study. 21

9 Use-case diagram of the system. 23

10 Context-free diagram of the system. 24

11 Data type of each variable before data preprocessing. 26

12 Data type of each variable after data preprocessing. 27

13 Correlation matrix of the features. 27

14 Landing page of Toxicheck. 35

15 Input form page of Toxicheck. 35

16 Results view of Toxicheck. 36

List of Tables

1 Data dictionary of the study. 20

2 Selected optimal features per classifier model. 28

3 Class distributions for train and test set. 29

4 Unscaled data samples for GA-RF. 30

5 Scaled data samples for GA-RF. 30

6 Hyperparameter tuning results per hybrid algorithm model. 31

7 Summary of performance metrics for batch 1 and batch 2. 32

8 Summary of performance metrics of models with feature scaling. . . . 33

9 Summary of performance metrics of models with all applied. 34

I. Introduction

A. Background of the Study

Nanomaterials (NMs) consist of particles whose size ranges between 1-100 nanometers on at least one structural dimension. Quantum effects can be employed to influence their physical and chemical properties due to their incredibly small size and high surface area to mass ratio [1]. Their physicochemical properties allow them to be ideal candidates for creating sustainable products in different application fields such as medicine, energy, and manufacturing industries.

At the nanoscale, almost every chemical can be altered to possess distinctive qualities that appeal more over large-sized materials [2]. This concept eventually led to the increased production of engineered nanomaterials (ENMs) as they posed better practicality of use and displayed improved physicochemical features compared to their respective conventional counterparts. However, despite the numerous beneficial effects ENMs provide, recent studies have shown that their unique properties also account for the risks they pose to human health and the environment.

According to [3], the health hazards and risks of nanoparticles (NPs) are linked to their physicochemical features. Due to their minute structure, ENMs can penetrate through cell membranes and cause intracellular and genetic damage. Additionally, ENMs can also create toxic mixtures and compounds with other chemicals [4]. With the continued use and production of these nanomaterials, the possibility of exposure to them becomes a major concern. To avoid adding to this issue, ENMs must be assessed before they are mass-produced to determine which of them are toxic and what features they have that contribute to their toxicity.

Past efforts in studying nanotoxicology majorly involve in-vitro and in-vivo experiments but both approaches require live samples and safety regulations due to the nature of their tests. In-silico testing, on the other hand, involves performing simulations and developing predictive models using computers and machine learning (ML) algorithms. It eliminates the need for animal testing, lowers the cost and time, and enhances prediction power [5].

In-silico methods are usually centered around Quantitative Structure-Activity Relationship (QSAR) which is an effective computational technique to ascertain the relationship between a particle's properties and its activity. It is built on the assumption that biological effects are connected to a compound's chemical structure and physicochemical characteristics [2]. QSAR is already widely used in developing predictive models across various studies involving toxicity classification however, initial attempts were limited due to small datasets [6].

Recently, there's been an increasing trend of implementing hybrid algorithms on classification problems. The principle of using this approach is to use multiple simple algorithms together to complement each other and increase the accuracy of the model by resolving problems that the other cannot solve on its own. For instance, the study done by [7] used two distinct machine learning algorithms for feature selection and as a classifier in a sequential pattern to produce improved results for Alzheimer's disease classification.

In this study, nano-QSAR models are developed using hybrid algorithms to classify the toxicity of nanomaterials. These models provide a safer risk assessment approach for ENMs and NPs and are used as guides for manufacturing safe-by-design NPs in the future.

B. Statement of the Problem

No studies using hybrid algorithms for nano-QSAR models were published yet at the time of this paper's making. Thus, the researcher wants to investigate the performance of hybrid algorithm models when applied to nanomaterial toxicity classification.

C. Objectives of the Study

This research aims to develop a predictive model for nanomaterial toxicity classification based on hybrid algorithm using cell viability as a toxicity measure and integrate it into a web application that will serve as an in-silico testing tool for researchers and professionals. It is directed toward the following objectives:

1. Use Genetic Algorithm (GA) for feature selection to identify relevant physico-chemical features that affect nanomaterial toxicity.
2. Split the dataset where 80% is for training and 20% is for testing.
3. Apply SMOTE to address the class imbalance.
4. Use StandardScaler to apply feature scaling on numerical values.
5. Use GridSearch for Hyperparameter Tuning of classifier algorithms.
6. Use Logistic Regression (LR), Artificial Neural Network (ANN), and Random Forest (RF) as classifiers in model training.
7. Assess each model's performance using the following metrics: accuracy, precision, recall, F1 score, ROC-AUC, and MCC.
8. Develop a web-based application built on the best-performing model to provide a toxicity classification of a nanomaterial described by the user input.
    (a) The application has an input panel where the users can enter values about the nanomaterial and the cell assay's properties.
    (b) The application saves the user's input when the submitted form is valid and it has a reset button that clears the saved values on the form.
    (c) The application has an output panel where the model's metrics are displayed.

(d) The application prints the classification result upon a valid submission. It will print toxic or non-toxic depending on the model's prediction.

D. Significance of the Project

Using a large dataset containing features of nanomaterials from different groups, the model can classify whether a nanomaterial is toxic or not regardless of its type. Therefore, its application extends to classifying not just metal oxides, but also other types such as carbon-based and nanocomposites. The study can provide helpful insights into what physicochemical attributes have a particular effect on the toxicity of nanomaterials using feature selection. This can aid nanomaterial manufacturers to create NPs that are safe by design and reduce the production of novel ENMs that are toxic. Consequently, it reduces the need to study new toxic ENMs and the mixtures they can produce.

Additionally, this study investigates if the hybrid algorithms used in creating the models can yield strong predictive powers relative to already existing nano-QSAR models for toxicity classification. It provides a new in-silico approach to assessing the risk of nanomaterials by using a machine learning model that has not been tested yet in the field. It gives a safer option for testing potentially toxic nanomaterials using computer simulations instead of in-vivo and in-vitro experiments.

E. Scope and Limitations

The study focuses on building toxicity classification models using the dataset gathered from meta-analysis done by [8]. Specifically, the scope and limitations of the study are as follows:

1. The dataset used is obtained from assessing 93 peer-reviewed articles. It has 2896 individual data points consisting of 16 predictors and 1 response variable.

2. The endpoint of this study is cell viability which is a binary variable where 0 means non-toxic ($> 50\%$ cell viability) and 1 means toxic ($\leq 50\%$ cell viability).

3. The nano-QSAR models only use the following machine learning algorithms: Genetic Algorithm, Logistic Regression, Artificial Neural Network, and Random Forest.

4. The web-based application can only produce predictions using the features that are included in the dataset: NP type, diameter, concentration, interference, colloidal stability, positive control, cell name, cell culture, cell type, cell morphology, cell age, cell source, test, test indicator, biochemical metric, and exposure time.

F. Assumptions

The following are assumptions on the web application:

1. All fields are filled before submission.

2. The user does not input invalid values.

II. Review of Related Literature

The unique physicochemical properties of nanomaterials make them suitable for manufacturing new products that possess better efficiency in specific and targeted activities. These distinctive benefits they provide give them an edge against the conventional larger-sized materials [2].

However, recent studies have shown that the utilization of nanomaterials can pose hazards to health and the environment due to potential toxicity linked to their properties [3]. Continued widespread use of these materials in industries, therefore, makes exposure to them unavoidable. Various testing approaches were conducted to assess the risks of nanomaterials, starting from experimental studies that required live samples and cell cultures to operate. Eventually, improved computational methods were adapted and allowed for machine learning algorithms to be the basis of assessing the toxicity of nanomaterials.

Using novel models derived from a selection of machine learning techniques, newer approaches can be discovered and evaluated based on their performance on classification problems. In particular, this study aims to use hybrid algorithms to develop a Quantitative Structure-Activity Relationship (QSAR) model for predicting the toxicity of different nanomaterials.

A. Nanomaterials

Nanomaterials (NMs) consist of particles whose size ranges between 1-100 nanometers on at least one structural dimension. Due to their minute size and high surface area to mass ratio, quantum effects can be employed to influence their physical and chemical properties [1]. This principle consequently allows them to behave differently than larger materials and exhibit desirable physicochemical properties that are ideal for creating sustainable products in different application fields such as medicine, energy, and industries. However, it also renders them to be unpredictable at times, as nanomaterials can undergo drastic changes to their properties with the slightest change in particle size.

At the nanoscale, almost every chemical can be altered to possess distinctive qualities that appeal more over their conventional bulk counterparts [2]. This eventually inspired the widespread use of engineered nanomaterials (ENMs) as they posed better practicality of use on material innovation. ENMs allowed the production of improved products by solving the flaws that otherwise would exist on a material without any NM. Some notable innovations include water-proof textiles, self-cleaning plastics, and cleaner pesticides.

However, despite the numerous beneficial effects ENMs provide, recent studies have shown that the unique properties that give them the ability to enhance materials also account for the risks they pose to human health and the environment. According to [3], the health hazards and risks of nanoparticles (NPs) are linked to their physicochemical features. Because of their nanoscopic size, ENMs can enter the human body through multiple pathways such as ingestion, inhalation, absorption via skin, and direct injection for therapeutic purposes [9]. Once inside the system, they can penetrate through cell membranes and cause intracellular damage as well as harm organs depending on the degree of exposure.

NMs may also interact with other contaminants to create a mixture of compounds after being released into the environment [4]. These new incidental nano-mixtures are recommended to be assessed on their own as these substances possess altered structures that may exhibit toxic properties. Additionally, toxicological studies show that NMs may have an impact on aquatic species and unicellular aquatic organisms (e.g., *Daphnia magna*) as substantiated by higher mortality rates and in-vitro assessment results.

Considering the extensive use and manufacture of nanomaterials, the possibility of exposure to them becomes unavoidable and a major concern. To avoid adding on and complicating this issue, ENMs must be assessed before they are mass-produced to determine which of them are toxic and what features they have that contribute to their toxicity.

B. Testing Approaches

As the field of nanotechnology rapidly grows, the hazard aspect of this subject is explored thoroughly through various testing approaches to assess the risk of nanomaterials. Past toxicological studies include experimental setups that primarily involved the use of in-vitro and in-vivo techniques. These methods were used because they describe the biological effects of nanomaterials on cells.

In-vitro assays take place in a controlled environment, usually in a petri dish or a test tube, outside of a living organism [10]. It uses cultures and isolated cells to perform assessments on a cellular and molecular level. This approach offers the benefits of being cost-effective, time-efficient, and not requiring animal use. However, one major disadvantage of in-vitro testing is samples fail to replicate the mechanistic functions of a whole organism, which in turn, can lead to unreliable and inaccurate results. In-vivo experiments, on the other hand, make use of live samples [10]. It is appropriate for studying the overall effects of a treatment on a living organism because it addresses the complexity of organ systems. With its high translatability to human systems, it can provide better evaluations of the toxicity of nanomaterials to health. However, this poses unethical issues when causing distress and discomfort to the animals that are used for testing.

Eventually, improved computational methods were adapted that allowed for machine learning algorithms to be the basis of assessing the toxicity of nanomaterials. In-silico testing involves performing simulations and developing predictive models using computers. It eliminates the need for animal testing, lowers the cost and time, and enhances prediction power [5]. Not only do these in-silico methods provide a faster and better alternative than the past methodologies [11], but they can also identify the physicochemical features that make a particular nanomaterial toxic. By doing feature selection and analyzing feature importance, these models can pinpoint which properties have significant influences on toxicity. Additionally, they can be used side by side with in-vivo and in-vitro models to fix the gaps in their procedures and yield more reliable outcomes.

In-silico methods are usually centered around Quantitative Structure-Activity Relationship (QSAR) which is an effective tool to ascertain the relationship between a particle's properties and its activity. It is built on the assumption that biological effects are connected to a compound's chemical structure and physicochemical characteristics [2]. However, these models require to be validated and regulated first and this is done by applying the OECD principles as guidelines.

QSAR is already widely used in developing predictive models across various studies involving toxicity classification. However, past attempts at toxicity classification were limited by small datasets [6]. This emphasizes the need for further research using larger datasets to fully explore the potential of machine learning in this field.

C. Machine Learning

A study done by [2] created an in-silico model of an in-vivo experiment using 6 machine learning algorithms to predict the toxicity of metallic nanomaterials on *Daphnia magna*. The study found that the random forest, artificial neural network, and k-nearest neighbor models displayed the best performance, but this was only marginally better compared to the other models. Furthermore, it has been suggested through the feature importance analysis that molecular descriptors and physicochemical properties were generally significant within most models [2]. However, some features related to exposure conditions produced slightly conflicting results.

Another study done by [12] involved the use of an in-silico model for predicting the interaction of TiO2-based nano-mixtures to *Daphnia magna*. In their study, they used random forest algorithm which yielded an R^2 value equal to 0.9928. It showed better predictive performance compared to existing CA and IA models and is suitable to be a low-cost alternative for assessing the risk of TiO2-based nano-mixtures.

A decision-tree-based KDD process was developed by [13] for predictive modeling of AgNPs-induced toxicity. This study yielded high f-score measures on classifying these nanoparticles for both the toxic (93.1%) and the nontoxic class (98.3%). [14], on the other hand, proposed 4 models using SMILES-based optimal descriptors and MC-PLS modeling to characterize the toxicity of 21 metal oxide nanoparticles on A549 cells. Their 4 models resulted in a high R^2 score of 0.8, indicating reliability, stability, and satisfactory predictive ability. Other studies have also utilized ensemble [5], simple rule-based model via association rule mining [3], and partial least square regression [6].

Although these models have generated satisfactory results in terms of predicting and classifying the toxicity of nanomaterials and nanoparticles, their findings only apply to certain groups of NPs that are included in the respective datasets the researchers used. Due to limited information and a small volume of data points, these models require further validation from newer models that are trained with larger and more general datasets.

The heterogeneities of published literature in terms of data quality can also be an issue. However, this was later addressed by [15] by using data gap filling and PChem score-based screening approaches to improve the completeness of the extracted datasets. They built models using datasets with different attribute combinations [15]. It yielded 93% as the highest F1 score (Dose + PChem + Tox) while 78% as its lowest (Dose + Tox). Results show that by adapting these two preprocessing techniques, a meta-analysis of nanotoxicity literature can also be an innovative alternative for the risk assessment of nanomaterials.

Recently, there's been an increasing trend of implementing hybrid algorithms on classification problems. The principle of using this approach is to use multiple simple algorithms together to complement each other and increase the accuracy of the model by resolving problems that the other cannot solve on its own. For instance, the study done by [7] used a machine learning technique for feature selection and a separate machine learning technique as a classifier. This study adapts this technique to investigate its efficacy in the field of nanotoxicology.

By using genetic algorithm for feature selection and combining it with an ML classifier (LR, ANN, RF), the study aims to develop nano-QSAR models using hybrid algorithm with cell viability as its well-defined endpoint. A dataset containing 2896 individual points of NPs varying in type is used to train this model to expand the model's applicability domain.

III. Theoretical Framework

A. Nanomaterial Toxicity

Nanomaterial toxicity refers to the potential harmful effects that nanoscale materials may have on living organisms and the environment[16]. These materials have unique physical and chemical properties that can make them more toxic than their bulk counterparts [2]. Some potential effects of nanomaterial toxicity include damage to DNA, oxidative stress, and inflammation. However, the toxicity of nanomaterials can vary depending on factors such as the type of material, its size, shape, and surface properties, and the method of exposure [3]. Due to the innate volatile nature of nanomaterials, further research is needed to fully understand the potential risks associated with them.

B. In-Silico Toxicological Testing

In-silico toxicological testing is a method of evaluating the potential toxicity of a substance using computer-based models and simulations. This approach is used to predict the potential effects of a substance on living organisms, without the need for laboratory experimentation such as in-vivo and in-vitro testing. [10]. In-silico methods are becoming increasingly popular in toxicology due to the speed, cost-effectiveness, and ability to predict the toxicity of large numbers of chemicals quickly [11].

B.1. Quantitative Structure-Activity Relationship

In-silico methods are usually focused on using Quantitative Structure-Activity Relationship (QSAR) models. It is a computational method used to predict the toxicity of a chemical based on its molecular structure [2]. These models are mathematical equations that are derived from a set of known toxicological data and are used to determine the toxicity of new chemicals. It is essential for it to be trained using a large dataset to enhance its accuracy [6]. There are four main phases in the QSAR modeling workflow: data gathering, data preprocessing, modeling, and post-analysis.

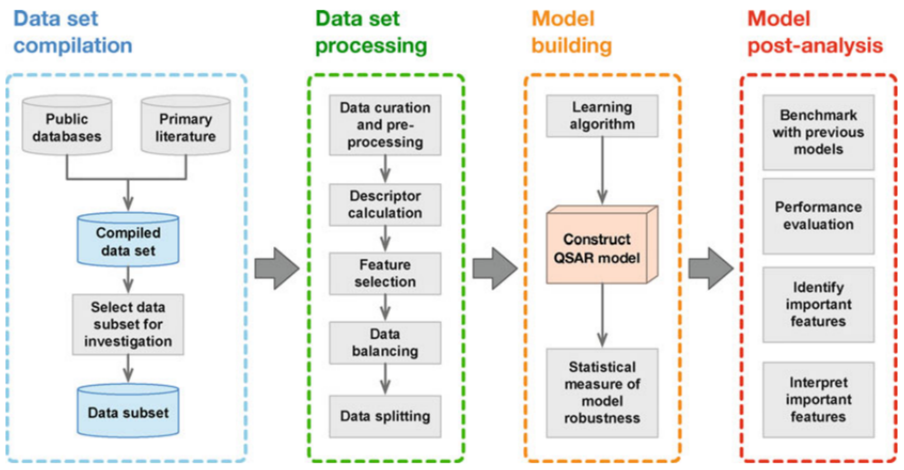


Figure 1: Schematic representation of the QSAR modeling workflow.

C. Synthetic Minority Oversampling Technique (SMOTE)

SMOTE is an oversampling technique used to handle imbalanced datasets in machine learning [17]. It is used to balance the class distribution by creating synthetic examples of the minority class. SMOTE works by selecting samples from the minority class and generating synthetic samples along the line segments connecting the selected sample to its k-nearest neighbors. The synthetic samples are added to the original dataset, increasing the frequency of the minority class. The number of neighbors can be adjusted to control the degree of oversampling. The higher its value, the more similar the synthetic samples are to the original samples.

D. Feature Scaling

Feature Scaling is a method used to standardize the range of independent variables or features of a data set to improve the performance of algorithms. The study used

StandardScaler() for feature scaling where numerical values are transformed to have a mean of 0 and a standard deviation of 1. This is to remove the mean and scale each feature to unit variance.

E. Hyperparameter Tuning

Hyperparameter tuning is the process of choosing a machine learning model's hyperparameters' ideal values. Hyperparameters are variables that are set before training the model to produce the optimal result from the model. They regulate how the learning algorithm behaves and they have a significant influence on the model's functionality and generalizability. The study used 5-fold cross-validation on GridSearchCV() alongside different parameter spaces that are unique to each model, to obtain the optimal hyperparameter values.

F. Machine Learning

Machine learning is a subfield of artificial intelligence that focuses on machines replicating human behavior. It creates models that can access data and use it to learn and provide data-driven output. The machine learning method begins with monitoring the data and searching for patterns. Then, the machine will generate conclusions and decisions based on the data provided. Recently, there has been an increasing trend of combining these different machine learning techniques for solving problems. This is known as hybrid algorithm. It makes use of the strengths of the combined algorithms to complement each other and overcome their limitations [18]. This includes using separate machine learning techniques for feature selection and classification and then merging their results together sequentially to create a model.

F.1. Feature Selection

Feature selection is the process of choosing a subset of relevant features for use in model construction. The goal of feature selection is to improve the accuracy and interpretability of the model by reducing the dimensionality of the input data. This can be done by removing irrelevant or redundant features, or by selecting a subset of the most informative features. This study will use genetic algorithm (GA) for feature selection.

Genetic Algorithm (GA)

GA is a method for optimization that is inspired by the process of natural selection [19]. It starts with a population of candidate solutions and iteratively applies genetic operators such as selection, crossover (recombination), and mutation to evolve the population toward an optimal solution.

The selection operator favors solutions that have higher fitness, which is a measure of how well the solution solves the problem at hand. The crossover operator combines the genetic information of two solutions to create new solutions. The mutation operator randomly changes the genetic information of a solution. If the fitness criterion is met, then the subset of features in that iteration is selected as the optimal solution.

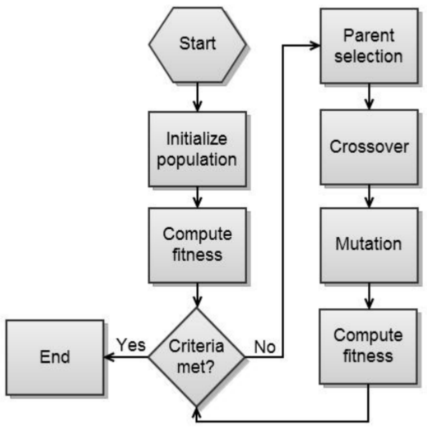


Figure 2: Genetic algorithm workflow.

F.2. Classifiers

Logistic Regression (LR)

LR is a statistical approach for assessing a dataset in which one or more independent variables affect a dichotomous result [20]. It predicts a binary result (represented by 0 and 1) from a set of independent variables. LR is represented by a linear equation and a sigmoid function that converts the linear equation's output to a probability value between 0 and 1. The equation takes the form of the log odds of the outcome variable, which is calculated using the input features and a set of parameters. The logistic regression algorithm estimates the parameters of the model by maximizing the likelihood of the observed data [20].

Artificial Neural Network (ANN)

ANN is a computational model that imitates the structure and functions of a human brain to be able to make decisions on classification and prediction [21]. It has the ability to learn and to create reasonable generalizations even on inputs that it has not been thought how to deal with. It is composed of interconnected nodes known as "neurons" which are connected by weighted edges that are fixed in each iteration. The ANN function is a deterministic calculation that describes the weighted sum from each neuron plus a bias.

Random Forest (RF)

RF is a decision tree-based machine learning algorithm that creates an ensemble of individual decision trees that run in parallel with each other [22]. The results are derived by calculating the average of all the decision tree values. To avoid compromising the robustness and the accuracy of the model's overall prediction, it is important that the individual decision trees have a low correlation to each other.

G. Performance Metrics

G.1. Accuracy

The accuracy metric describes the percentage of the prediction that the model classified correctly. It is defined by the following formula where TP = True Positive, TN = True Negative, FP = False Positive, and FN = False Negative.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

Figure 3: Accuracy rate formula.

G.2. Precision

Precision is the proportion of true positives over the total number of positive predictions made by the model. A high precision value means that the model is not producing many false positives. It is used in conjunction with the Recall metric for computing the F1 Score. It is defined by the following formula:

$$Precision = \frac{TP}{TP + FP}$$

Figure 4: Precision rate formula.

G.3. Recall

Recall is the proportion of true positives predicted by the model over the total actual positive instances in the dataset. A high recall value means that the model is able to correctly identify most of the positive instances. It is used in conjunction with the Precision metric for computing the F1 Score. It is defined by the following formula:

$$Recall = \frac{TP}{TP + FN}$$

Figure 5: Recall rate formula.

G.4. F1 Score

The F1 Score is the harmonic mean of precision and recall. A high F1 score indicates that the model has a good balance between precision and recall. It is defined by the following formula:

$$F1\ Score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

Figure 6: F1 score formula.

G.5. Receiver Operating Characteristic (ROC)

The ROC curve is a graph that plots the True Positive Rate (TPR) and the False Positive Rate (FPR) across different discrimination thresholds. It is a way to evaluate the performance of a classifier, thus, it can also be used to compare different classifiers. It is used in conjunction with the area under the curve (AUC) metric.

G.6. Area Under the Curve (AUC)

AUC is a scalar value that summarizes the ROC curve. Its value ranges between 0 and 1, where 1 represents a perfect classifier and a score of 0.5 represents a random classifier. It is independent of the classification threshold, so it is useful for classifiers that do not have a nonlinear decision boundary.

G.7. Matthew's Correlation Coefficient (MCC)

MCC is a measure of the performance of a binary classifier, specifically designed to handle imbalanced datasets. It ranges from -1 to 1, where 1 represents a perfect prediction, 0 represents no better than a random prediction, and -1 represents total disagreement between prediction and observation. This will be the main metric to determine the best-performing model. It is defined by the following formula:

$$MCC = \frac{TN \cdot TP - FN \cdot FP}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

Figure 7: Matthew's correlation coefficient score formula.

IV. Design and Implementation

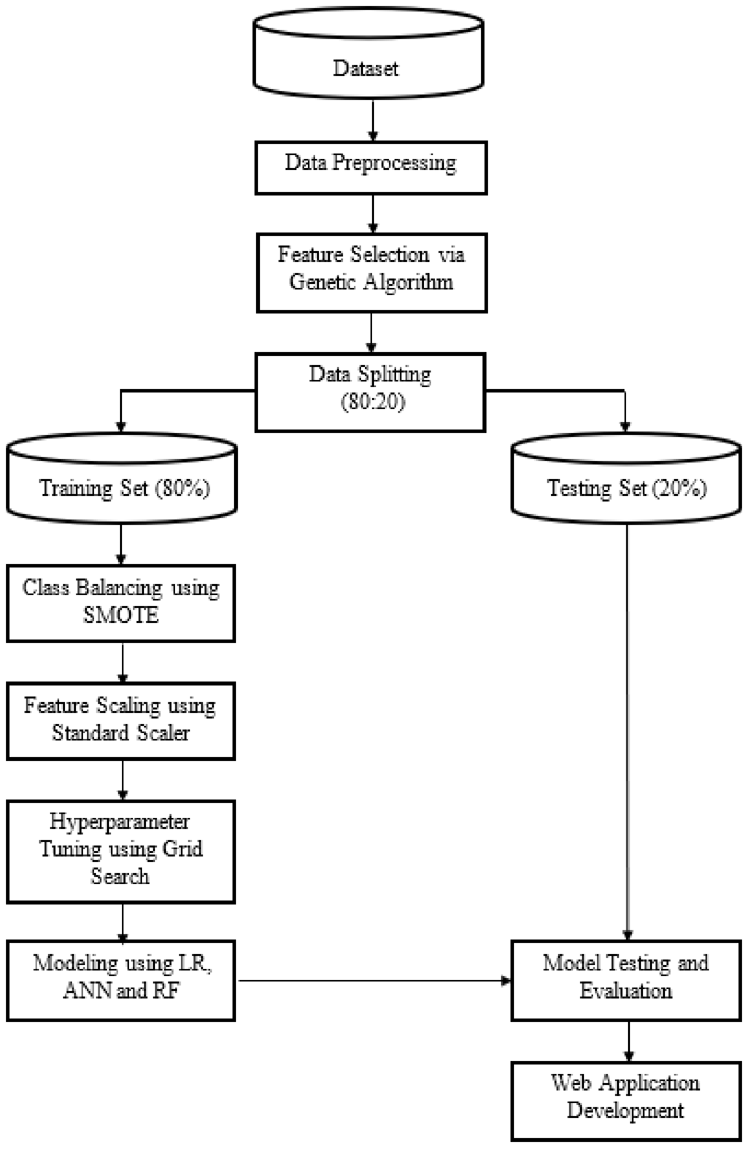
A. Dataset

The dataset of this study focuses on building nano-QSAR models using a publicly available dataset collated by [8] on their meta-analysis, in which they assessed a total of 93 peer-reviewed articles about nanomaterial cytotoxicity. It yields 2896 individual data points consisting of 17 features: 16 predictor variables and 1 response. The predictor variables consist of Nanoparticle-related features (NP type, diameter, concentration, interference, colloidal stability, and positive control); Cell-related features (cell name, cell culture, cell type, cell morphology, cell age, and cell source); and methodological parameters (test, test indicator, biochemical metric, exposure time) [8]. The response variable is cell viability which is a binary variable where 0 is labeled as non-toxic ($> 50\%$ cell viability) and 1 is labeled as toxic ($\leq 50\%$ cell viability). The dataset obtained is divided into the training and testing set on an 80:20 ratio.

Table 1: Data dictionary of the study.

Variable	Data Type	Values
NP Type	Binary	0 Organic – 1 Inorganic
Diameter	Numerical	1 to 957 (nm)
Concentration	Numerical	0 to 15000 (μm)
Interference Check	Binary	0 No – 1 Yes
Colloidal Stability Check	Binary	0 No – 1 Yes
Positive Control	Binary	0 No – 1 Yes
Cell Name	Categorical	81 Unique Values
Cell Type	Binary	0 Human – 1 Animal
Cell Culture	Binary	0 Primary Cell – 1 Cell Line
Cell Morphology	Categorical	15 Unique Values
Cell Age	Binary	0 Adult – 1 Embryonic
Cell Source	Categorical	30 Unique Values
Test	Categorical	21 Unique Values
Test Indicator	Categorical	18 Unique Values
Biochemical Metric	Categorical	26 Unique Values
Exposure Time	Numerical	1 to 336 (hours)
Cell Viability	Binary	0 Not Toxic – 1 Toxic

B. General Workflow



Dataset
Data Preprocessing
Feature Selection via Genetic Algorithm
Data Splitting (80:20)
Training Set (80%)
Testing Set (20%)
Class Balancing using SMOTE
Feature Scaling using Standard Scaler
Hyperparameter Tuning using Grid Search
Modeling using LR, ANN and RF
Model Testing and Evaluation
Web Application Development

Figure 8: General workflow of the study.

B.1. Data Preprocessing

The dataset contains String-type binary and categorical variables. In order to prepare the binary data for analysis, LabelEncoder() is used to convert these values into an integer type (0 or 1). On the other hand, the remaining categorical features are transformed into binary variables using One-Hot encoding. Additionally, rows with missing and invalid values are dropped from the analysis. Once the dataset is processed, the data is divided into predictors and response to prepare it for feature selection.

B.2. Feature Selection using Genetic Algorithm

The predictor's dimensionality is reduced using genetic algorithm. The list of optimal features is produced after a 5-fold cross-validation run of GeneticSelectionCV() from sklearn-genetic library. Once the optimized subset of features is selected, the dataset is split into a training set (80%) and a testing set (20%). This ratio is done to avoid overfitting and to preserve the integrity of the testing data that will be used for evaluation purposes later on.

B.3. Model Implementation

Since the dataset has a class imbalance that may hinder the model's prediction power, Synthetic Minority Oversampling Technique (SMOTE) is applied. SMOTE is a wildly used resampling technique to create synthetic samples that represent the minority class and balance the ratio of classes. Furthermore, since the numerical features in the dataset (diameter, concentration, exposure time) have a wide range of values, feature scaling is applied using StandardScaler() to rescale the data to have a mean of 0 and a standard deviation of 1.

The training set is used in building the models for LR, ANN, and RF as classifiers. To improve the results of these algorithms, hyperparameter tuning is used for each model. The hyperparameter spaces of each model are passed to GridSearchCV() and are executed with 5-fold cross-validation. The resulting models from GridSearch are used to predict the toxicity of the data points from the testing set.

B.4. Model Evaluation and Application

Once results are recorded, the performance of these models is compared to each other and is evaluated using the following metrics: accuracy, precision, recall, F1 score, ROC-AUC curve, and MCC. Using the MCC metric, the best-performing model is selected to be integrated into the web-based application.

Lastly, a web-based application is developed based on the best-performing model from the study. The application is a system that allows users to enter information about the properties of a nanomaterial and the cell-based assay to be exposed to the nanomaterial. It displays the result of whether that nanomaterial is toxic or non-toxic based on the cell viability of the described assay.

C. Use-Case Diagram

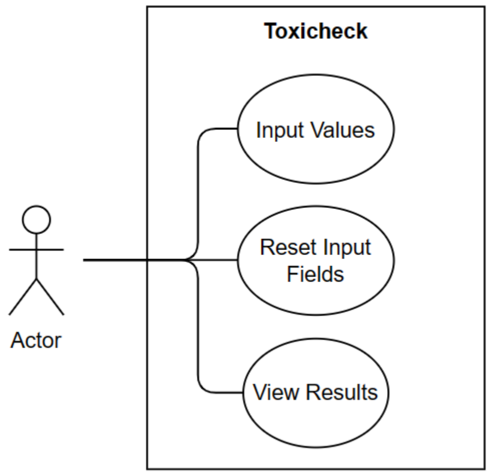


Figure 9: Use-case diagram of the system.

Figure 9 shows the functionalities that an end user has access to with the system. The user can input information about a nanomaterial and the cell assay's properties into the system by filling out all the available fields in the web-based application. If the submitted form is valid, the input values are saved in the panel until the reset button is clicked. A summary of the result produced by the best-performing model is displayed along with the toxicity classification result (toxic or non-toxic) in bold text.

D. Context-Free Diagram

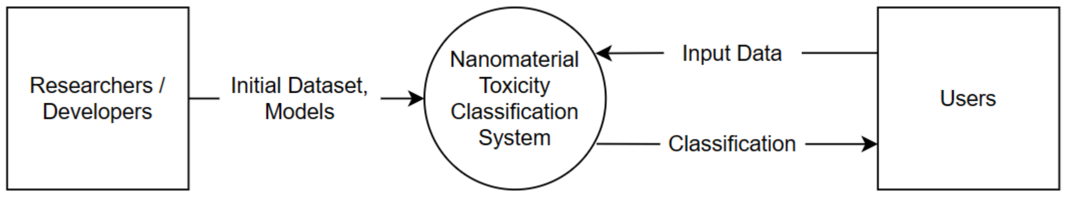


Figure 10: Context-free diagram of the system.

Figure 10 shows that there are two entities in the system. The initial data and the nano-QSAR models are entered into the system by the developer. The user enters the input data and the system processes this to predict and produce a result that is displayed back to the user as the output.

E. System Architecture

The web-based application is built and developed using the Python-based Django framework. The genetic algorithm uses `sklearn-genetic` while LR, ANN and RF use `scikit-learn`. Other packages include the following: `pandas`, `numpy`, and `imbalanced-learn` are used for handling data; `joblib` and `pickle` are used for saving pipelines into a file; `matplotlib` and `seaborn` are used for visualization.

1. Python 3.11.2
    (a) Django 4.1.7
    (b) imbalanced-learn 0.10.1
    (c) matplotlib 3.7.1

(d) numpy 1.23.1

(e) pandas 2.0.1

(f) scikit-learn 1.2.2

(g) seaborn 0.12.2

(h) sklearn-genetic 0.5.1

F. Technical Architecture

This application is currently deployed using localhost only. However, since this system is intended to be deployed as a web application, most of the computing is going to be done by the server. Therefore, minimal resources are required when it is used by the user. The minimum requirements for the application are as follows:

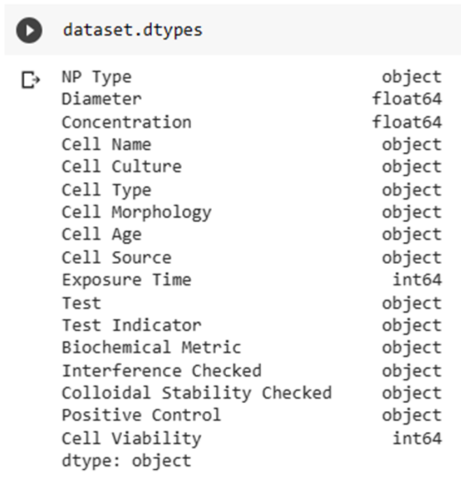
1. 2 GHz processor
2. 1 GB disk space
3. 2 GB of RAM
4. Any OS
5. Web browser

V. Results

A. Data Preprocessing

Exploratory Data Analysis

The dataset gathered from [8] has 16 predictors that describe its nanoparticle-related features, cell-related features, and methodological parameters related to the response variable, which is cell viability. The figure below shows the data type of each variable before data preprocessing.



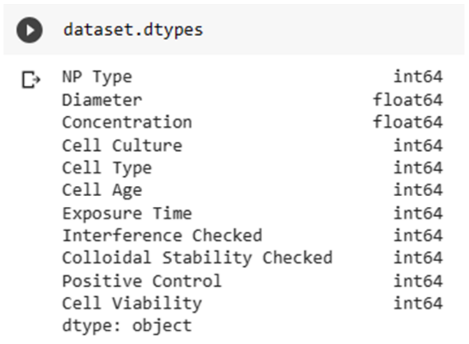
```
dataset.dtypes

NP Type                       object
Diameter                     float64
Concentration                float64
Cell Name                     object
Cell Culture                  object
Cell Type                     object
Cell Morphology               object
Cell Age                      object
Cell Source                   object
Exposure Time                  int64
Test                          object
Test Indicator                object
Biochemical Metric            object
Interference Checked          object
Colloidal Stability Checked   object
Positive Control              object
Cell Viability                 int64
dtype: object
```

Figure 11: Data type of each variable before data preprocessing.

Figure 11 shows that there are four numerical features in the dataset: diameter, concentration, exposure time, and cell viability. Both diameter and concentration are float-type decimals, while exposure time and cell viability are integers. The remaining 12 variables are String-type objects that are either binary or categorical, as shown in the data dictionary in Table 1.

The binary String-type variables are transformed to an integer type (0 or 1) using the LabelEncoder() function, while the remaining categorical features are dropped due to skill-related issues encountered while applying One-Hot encoding. There are no rows with invalid values, so the number of data points is retained.



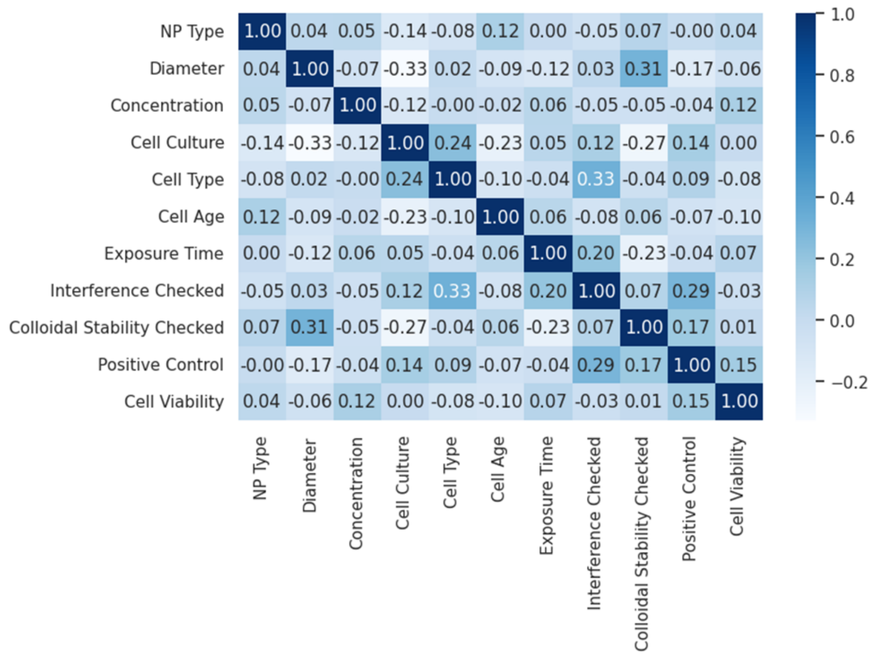
```
dataset.dtypes

NP Type                          int64
Diameter                       float64
Concentration                  float64
Cell Culture                     int64
Cell Type                        int64
Cell Age                         int64
Exposure Time                    int64
Interference Checked             int64
Colloidal Stability Checked      int64
Positive Control                 int64
Cell Viability                   int64
dtype: object
```

Figure 12: Data type of each variable after data preprocessing.

Figure 12 shows that there are 11 variables left after dropping the categorical values. Additionally, after the LabelEncoder() function is used, all of the remaining variables have numerical values that are either float or integer types.

The feature variables of this updated dataset are checked for possible correlations with each other. This is done to ensure that there is no multicollinearity between variables that can negatively impact the model's performance by creating bias and overfitting.



	NP Type	Diameter	Concentration	Cell Culture	Cell Type	Cell Age	Exposure Time	Interference Checked	Colloidal Stability Checked	Positive Control	Cell Viability
NP Type	1.00	0.04	0.05	-0.14	-0.08	0.12	0.00	-0.05	0.07	-0.00	0.04
Diameter	0.04	1.00	-0.07	-0.33	0.02	-0.09	-0.12	0.03	0.31	-0.17	-0.06
Concentration	0.05	-0.07	1.00	-0.12	-0.00	-0.02	0.06	-0.05	-0.05	-0.04	0.12
Cell Culture	-0.14	-0.33	-0.12	1.00	0.24	-0.23	0.05	0.12	-0.27	0.14	0.00
Cell Type	-0.08	0.02	-0.00	0.24	1.00	-0.10	-0.04	0.33	-0.04	0.09	-0.08
Cell Age	0.12	-0.09	-0.02	-0.23	-0.10	1.00	0.06	-0.08	0.06	-0.07	-0.10
Exposure Time	0.00	-0.12	0.06	0.05	-0.04	0.06	1.00	0.20	-0.23	-0.04	0.07
Interference Checked	-0.05	0.03	-0.05	0.12	0.33	-0.08	0.20	1.00	0.07	0.29	-0.03
Colloidal Stability Checked	0.07	0.31	-0.05	-0.27	-0.04	0.06	-0.23	0.07	1.00	0.17	0.01
Positive Control	-0.00	-0.17	-0.04	0.14	0.09	-0.07	-0.04	0.29	0.17	1.00	0.15
Cell Viability	0.04	-0.06	0.12	0.00	-0.08	-0.10	0.07	-0.03	0.01	0.15	1.00

Figure 13: Correlation matrix of the features.

If there are no highly correlated variables, the individual effect of each predictor on the response variable is easier to determine. An absolute correlation coefficient score greater than 0.7 indicates the presence of multicollinearity, which must be addressed before it gets passed to the model.

Figure 13 shows that all of the variables have a weak correlation with one another, with values ranging within [-0.33, 0.33]. The highest positive correlation is between cell type and interference checked with 0.33 while the highest negative correlation is between diameter and cell culture.

B. Feature Selection using Genetic Algorithm

For each classifier model (LR, ANN, RF), a list of optimal features is produced after a 5-fold cross-validation run of the GeneticSelectionCV() function from sklearn-genetic. The table below shows all the features that are selected for each classifier model.

Table 2: Selected optimal features per classifier model.

Predictors	GA-LR	GA-ANN	GA-RF
NP Type	✓		✓
Diameter	✓	✓	✓
Concentration	✓	✓	✓
Cell Type	✓	✓	✓
Cell Culture	✓		✓
Cell Age	✓	✓	✓
Exposure Time	✓	✓	✓
Interference Check			
Colloidal Stability Check		✓	✓
Positive Control	✓	✓	✓

Table 2 shows that for all classifier models, the interference check predictor is removed as a feature. Additionally, for logistic regression, the colloidal stability check is also deemed insignificant, while NP type and cell culture are removed for the artificial neural network. This means that GA-LR has 8 features, GA-ANN has 7 features, and GA-RF has 9 features after feature selection.

C. Data Splitting and Class Balancing using SMOTE

Upon analyzing the target variable of the dataset from [8], a class imbalance is observed between the non-toxic class and the toxic class. The non-toxic (0) class has 2209 values, while the toxic (1) class has 687 points.

The data is partitioned into an 80-20 split, where 80% is for the training set and 20% is for the testing set. The training set has 2316 values, where 1765 are non-toxic and 551 are toxic. While the testing set has 580 values, 444 are non-toxic and 136 are toxic. To balance the uneven class distributions during the model training, SMOTE is applied to the training set. Table 3 summarizes the split and the distribution of classes in each set.

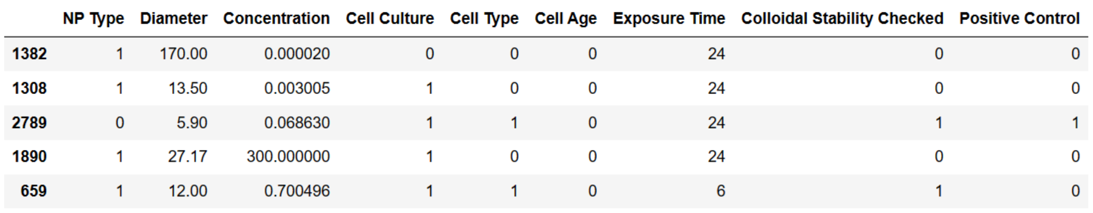
Table 3: Class distributions for train and test set.

	Non-Toxic	Toxic
Train Set	1765	551
Train Set (SMOTE)	1765	1765
Test Set	444	136

D. Feature Scaling

The numerical features in the dataset (diameter, concentration, exposure time) have a wide range of values, which causes higher variability in the data. The diameter has values ranging from 1 to 957 (in nm), the concentration ranges from 0 to 15000 in (µm), and the exposure time is from 1 to 336 (in hours)[8].

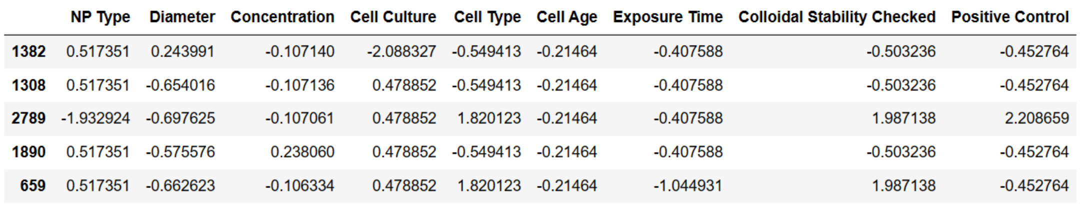
Table 4: Unscaled data samples for GA-RF.



	NP Type	Diameter	Concentration	Cell Culture	Cell Type	Cell Age	Exposure Time	Colloidal Stability Checked	Positive Control
1382	1	170.00	0.000020	0	0	0	24	0	0
1308	1	13.50	0.003005	1	0	0	24	0	0
2789	0	5.90	0.068630	1	1	0	24	1	1
1890	1	27.17	300.000000	1	0	0	24	0	0
659	1	12.00	0.700496	1	1	0	6	1	0

Using the StandardScaler() function, feature scaling is applied to all predictors, including the binary variables, to rescale both training sets (with and without SMOTE) to have a mean of 0 and a standard deviation of 1. Table 5 shows a sample of scaled training data to be used for GA-RF.

Table 5: Scaled data samples for GA-RF.

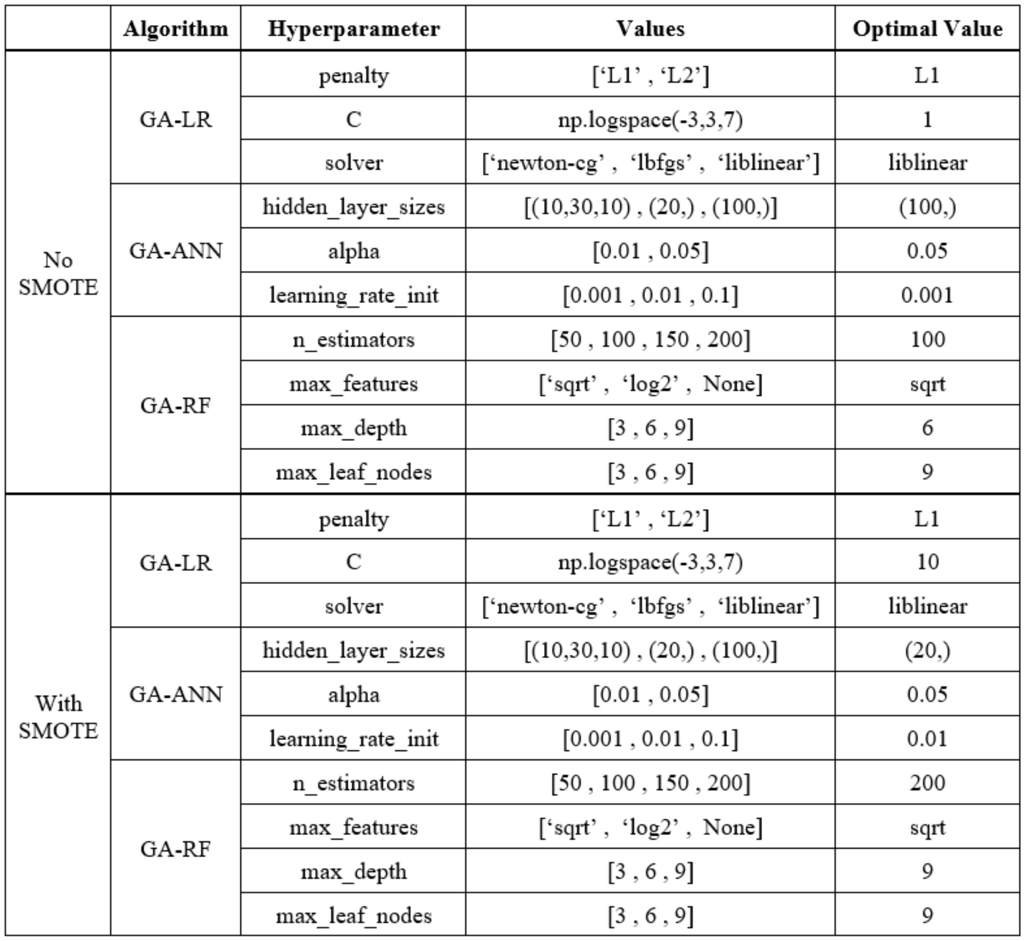


	NP Type	Diameter	Concentration	Cell Culture	Cell Type	Cell Age	Exposure Time	Colloidal Stability Checked	Positive Control
1382	0.517351	0.243991	-0.107140	-2.088327	-0.549413	-0.21464	-0.407588	-0.503236	-0.452764
1308	0.517351	-0.654016	-0.107136	0.478852	-0.549413	-0.21464	-0.407588	-0.503236	-0.452764
2789	-1.932924	-0.697625	-0.107061	0.478852	1.820123	-0.21464	-0.407588	1.987138	2.208659
1890	0.517351	-0.575576	0.238060	0.478852	-0.549413	-0.21464	-0.407588	-0.503236	-0.452764
659	0.517351	-0.662623	-0.106334	0.478852	1.820123	-0.21464	-1.044931	1.987138	-0.452764

E. Hyperparameter Tuning using Grid Search

After finalizing the training and testing sets from the previous steps, distinct hyperparameter spaces are declared for each classifier model. These spaces describe the values of each parameter to be tested using GridSearch to be able to obtain the combination of hyperparameter values that would yield the best performance.

Table 6: Hyperparameter tuning results per hybrid algorithm model.



	Algorithm	Hyperparameter	Values	Optimal Value
No SMOTE	GA-LR	penalty	[‘L1’ , ‘L2’]	L1
		C	np.logspace(-3,3,7)	1
		solver	[‘newton-cg’ , ‘lbfgs’ , ‘liblinear’]	liblinear
	GA-ANN	hidden_layer_sizes	[(10,30,10) , (20,) , (100,)]	(100,)
		alpha	[0.01 , 0.05]	0.05
		learning_rate_init	[0.001 , 0.01 , 0.1]	0.001
	GA-RF	n_estimators	[50 , 100 , 150 , 200]	100
		max_features	[‘sqrt’ , ‘log2’ , None]	sqrt
		max_depth	[3 , 6 , 9]	6
		max_leaf_nodes	[3 , 6 , 9]	9
With SMOTE	GA-LR	penalty	[‘L1’ , ‘L2’]	L1
		C	np.logspace(-3,3,7)	10
		solver	[‘newton-cg’ , ‘lbfgs’ , ‘liblinear’]	liblinear
	GA-ANN	hidden_layer_sizes	[(10,30,10) , (20,) , (100,)]	(20,)
		alpha	[0.01 , 0.05]	0.05
		learning_rate_init	[0.001 , 0.01 , 0.1]	0.01
	GA-RF	n_estimators	[50 , 100 , 150 , 200]	200
		max_features	[‘sqrt’ , ‘log2’ , None]	sqrt
		max_depth	[3 , 6 , 9]	9
		max_leaf_nodes	[3 , 6 , 9]	9

The list of optimal hyperparameter values is produced after a 5-fold cross-validation run of the GridSearchCV() function for each classifier. Table 6 shows all of the hyperparameters tested and their optimal values per model. The resulting models from GridSearch are used to predict the toxicity of the data points from the testing set.

F. Model Evaluation

F.1. Model Performance with SMOTE and Genetic Algorithm

Model training is divided into four batches. For the first batch, the classifiers are trained using the training set with SMOTE. The result of this batch determines whether SMOTE improved the results of the hybrid algorithms or not. For the second batch of runs, the classifiers are trained using the training set that is both not scaled and not resampled by SMOTE. This part compares the performance metrics of the hybrid classifiers with their base classifier counterparts. Table 7 summarizes the results from the first two batches of training.

Table 7: Summary of performance metrics for batch 1 and batch 2.

	Model	Accuracy	Precision	Recall	F1	AUC	MCC
No SMOTE	GA-LR	0.78	0.83	0.78	0.70	0.53	0.23
	GA-ANN	0.78	0.78	0.78	0.72	0.55	0.24
	GA-RF	0.80	0.82	0.80	0.74	0.57	0.33
With SMOTE	LR	0.59	0.72	0.59	0.62	0.61	0.19
	GA-LR	0.78	0.83	0.78	0.70	0.53	0.21
	ANN	0.67	0.76	0.67	0.69	0.68	0.30
	GA-ANN	0.77	0.72	0.77	0.72	0.56	0.19
	RF	0.71	0.76	0.71	0.73	0.68	0.33
	GA-RF	0.80	0.82	0.80	0.75	0.58	0.34

The result of batch 1 shows that applying SMOTE to GA-LR and GA-RF has a minimal effect on the performance metrics, with at most a 0.01% increase. However, GA-ANN receives a significant drop in precision and MCC scores after being resampled with SMOTE.

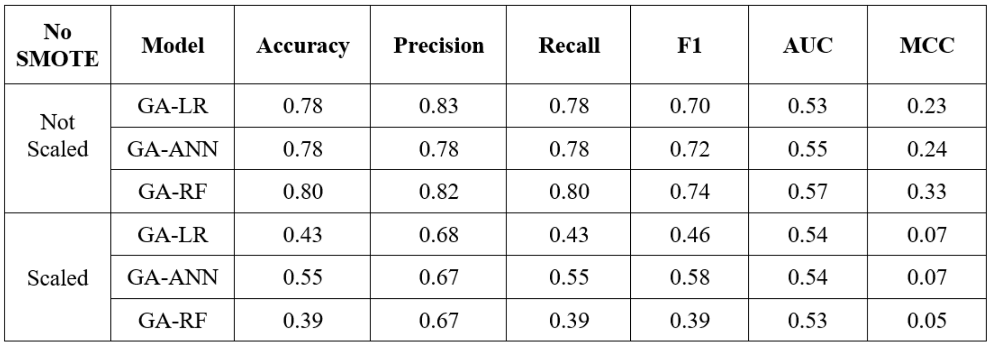
The result of batch 2 shows that there is a significant increase in the accuracy of the model from the base classifier to its hybrid counterparts after applying genetic algorithm for feature selection. With Logistic Regression increasing from 0.59 to 0.78, Artificial Neural Network increasing from 0.67 to 0.77, and Random Forest increasing from 0.71 to 0.80. However, it has a mixed effect on the other metrics, and there is a significant decrease observed in each model's AUC scores.

Each highest score per metric is highlighted as shown in Table 7, but the main metric used to determine which model is the best-performing model is the MCC score. Based on the results of the two batches, the best-performing model is GA-RF with SMOTE, with an MCC score of 0.34.

F.2. Model Performance with Feature Scaling

For the third batch of training, the classifiers are trained with the scaled train set. This part investigates whether feature scaling improves the performance metrics of the hybrid algorithm models. Table 8 shows the summary of the results of the models with feature scaling.

Table 8: Summary of performance metrics of models with feature scaling.



No SMOTE	Model	Accuracy	Precision	Recall	F1	AUC	MCC
Not Scaled	GA-LR	0.78	0.83	0.78	0.70	0.53	0.23
	GA-ANN	0.78	0.78	0.78	0.72	0.55	0.24
	GA-RF	0.80	0.82	0.80	0.74	0.57	0.33
Scaled	GA-LR	0.43	0.68	0.43	0.46	0.54	0.07
	GA-ANN	0.55	0.67	0.55	0.58	0.54	0.07
	GA-RF	0.39	0.67	0.39	0.39	0.53	0.05

Applying the StandardScaler() function for feature scaling decreases all of the performance metrics to a poor state, with scores reaching below 0.50 on most of the metrics. There is no new model that scored a higher MCC score than the current best-performing model identified from batch 2.

F.3. Model Performance with All Applied

For the last batch of training, the classifiers are trained with the scaled trained set with SMOTE. The result from this batch determines if applying all the techniques can improve the performance metrics of the models or not. Table 9 summarizes the performance metrics of the models after all techniques are applied.

Table 9: Summary of performance metrics of models with all applied.

With SMOTE	Model	Accuracy	Precision	Recall	F1	AUC	MCC
Not Scaled	GA-LR	0.78	0.83	0.78	0.70	0.53	0.21
	GA-ANN	0.77	0.72	0.77	0.72	0.56	0.19
	GA-RF	0.80	0.82	0.80	0.75	0.58	0.34
Scaled	GA-LR	0.48	0.65	0.48	0.51	0.52	0.03
	GA-ANN	0.49	0.65	0.49	0.52	0.52	0.03
	GA-RF	0.24	0.82	0.24	0.10	0.50	0.04

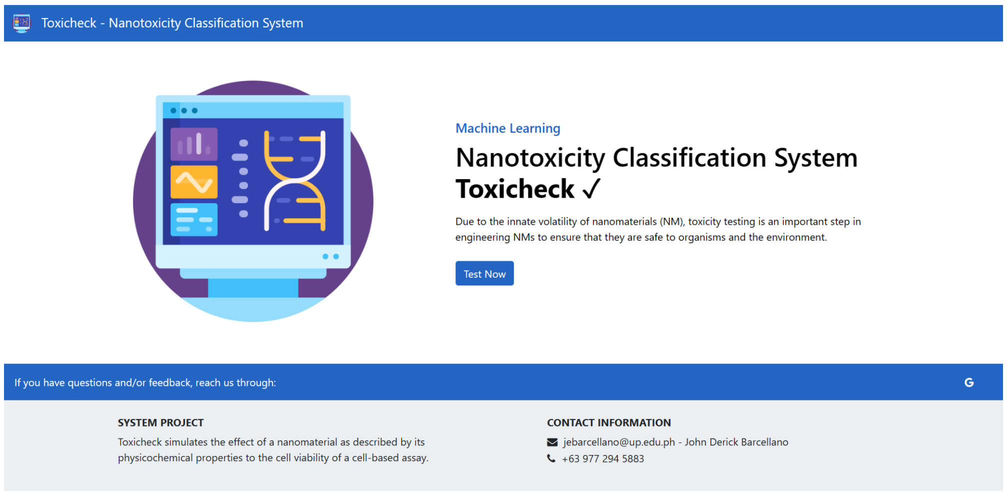
With similar effects observed in batch 3, applying the StandardScaler() function to the models has produced significantly lower metric scores across the table. With no improvements after using feature scaling, the best-performing model remains to be the GA-RF with SMOTE.

G. Web-Based Application

The web application, named Toxicheck, comprises two pages and one view: Landing Page, Input Form Page, and Results View. These pages are accessed through the designated buttons that are displayed on each page.

G.1. Landing Page

The landing page is the first page of the application that will be displayed to the user. It contains the title of the application, the system's logo, and a brief introduction to the system's purpose. Below the introduction, a button that says, "Test Now" can be clicked to redirect the user to the input form page.


Toxicheck - Nanotoxicity Classification System

Machine Learning

Nanotoxicity Classification System
Toxicheck ✓

Due to the innate volatility of nanomaterials (NM), toxicity testing is an important step in engineering NMs to ensure that they are safe to organisms and the environment.

Test Now

If you have questions and/or feedback, reach us through:

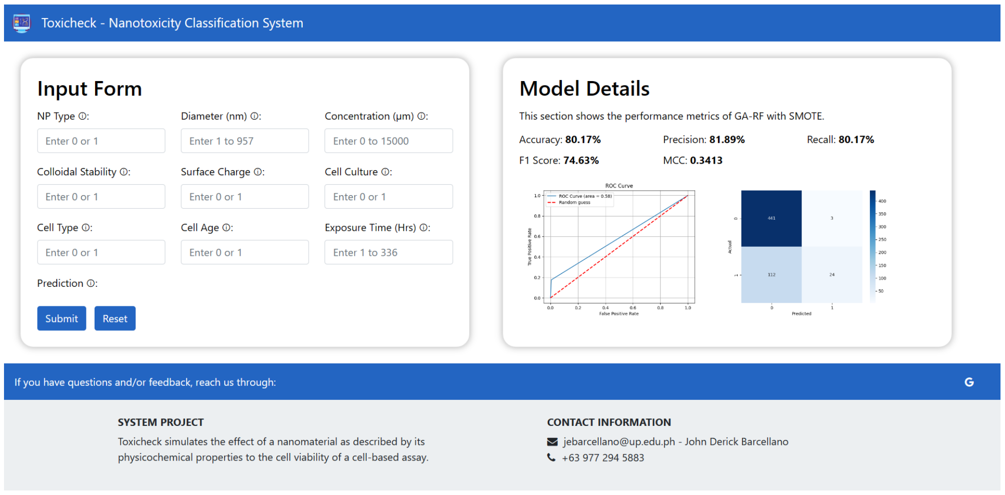
SYSTEM PROJECT
Toxicheck simulates the effect of a nanomaterial as described by its physicochemical properties to the cell viability of a cell-based assay.

CONTACT INFORMATION
jebarcellano@up.edu.ph - John Derick Barcellano
+63 977 294 5883

Figure 14: Landing page of Toxicheck.

G.2. Input Form Page

The input form page contains the navbar, two rectangular panels, and the footer. The navbar has the system's logo and name. If the user clicks this area, they will be redirected back to the landing page. The footer contains the contact information of the researcher, as well as a brief description of the system.


Toxicheck - Nanotoxicity Classification System

Input Form

NP Type ⓘ:	Diameter (nm) ⓘ:	Concentration (μm) ⓘ:
Enter 0 or 1	Enter 1 to 957	Enter 0 to 15000
Colloidal Stability ⓘ:	Surface Charge ⓘ:	Cell Culture ⓘ:
Enter 0 or 1	Enter 0 or 1	Enter 0 or 1
Cell Type ⓘ:	Cell Age ⓘ:	Exposure Time (Hrs) ⓘ:
Enter 0 or 1	Enter 0 or 1	Enter 1 to 336

Prediction ⓘ:

Submit Reset

Model Details

This section shows the performance metrics of GA-RF with SMOTE.

Accuracy: **80.17%** Precision: **81.89%** Recall: **80.17%**

F1 Score: **74.63%** MCC: **0.3413**

Actual \ Predicted	0	1
0	441	3
1	112	24

If you have questions and/or feedback, reach us through:

SYSTEM PROJECT
Toxicheck simulates the effect of a nanomaterial as described by its physicochemical properties to the cell viability of a cell-based assay.

CONTACT INFORMATION
jebarcellano@up.edu.ph - John Derick Barcellano
+63 977 294 5883

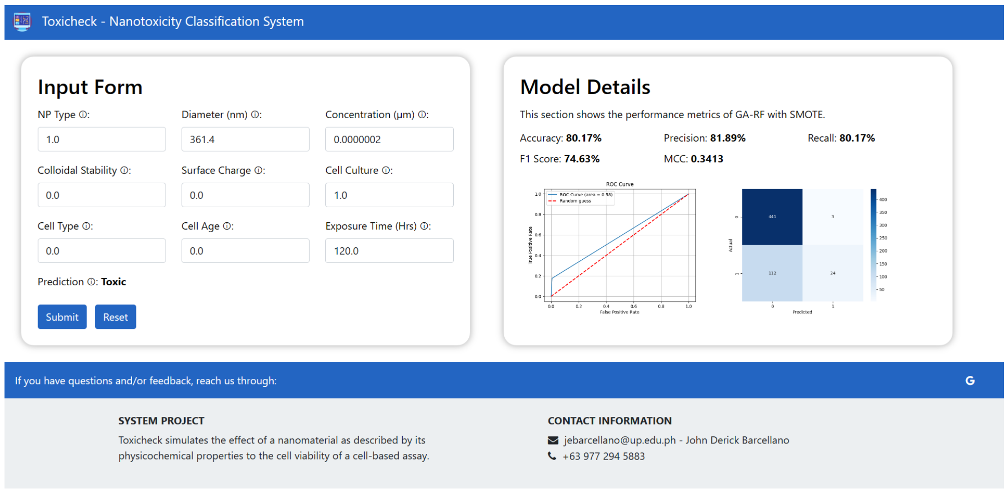
Figure 15: Input form page of Toxicheck.

The left panel of this page contains the input form and two buttons: submit and reset. This is where the input fields are displayed. If the user hovers over any of the field's names, a tooltip describing what the variable is about will show. The first five input fields are nanoparticle-related, while the remaining four are about the cell-based assay. Placeholder values are set to guide the user through the possible values that they can enter into the system. Once the form is completed, the user can click the 'submit' button to proceed to the results view.

The right panel contains the performance metrics of the model that is integrated into the system. It shows the accuracy, precision, recall, F1 score, MCC, the ROC-AUC curve, and the confusion matrix of GA-RF with SMOTE.

G.3. Results View

Once the form is submitted, the system redirects the user to the results view. This view retains all the functionalities described in the input form page. Additionally, the prediction field now displays the classification result (non-toxic or toxic) made by the model using the input values. The submitted values are saved on the input fields and are only cleared once the user clicks the 'reset' button.



Input Form

NP Type ⓘ: 1.0	Diameter (nm) ⓘ: 361.4	Concentration (μm) ⓘ: 0.0000002
Colloidal Stability ⓘ: 0.0	Surface Charge ⓘ: 0.0	Cell Culture ⓘ: 1.0
Cell Type ⓘ: 0.0	Cell Age ⓘ: 0.0	Exposure Time (Hrs) ⓘ: 120.0

Prediction ⓘ: **Toxic**

Model Details

This section shows the performance metrics of GA-RF with SMOTE.

Accuracy: **80.17%** Precision: **81.89%** Recall: **80.17%**

F1 Score: **74.63%** MCC: **0.3413**

Actual \ Predicted	0	1
0	441	3
1	112	24

If you have questions and/or feedback, reach us through:

SYSTEM PROJECT

Toxicheck simulates the effect of a nanomaterial as described by its physicochemical properties to the cell viability of a cell-based assay.

CONTACT INFORMATION

jebarcellano@up.edu.ph - John Derick Barcellano

+63 977 294 5883

Figure 16: Results view of Toxicheck.

VI. Discussions

This research aims to evaluate the performance of three hybrid algorithm-based models (GA-LR, GA-ANN, GA-RF) in predicting whether a nanomaterial is toxic or non-toxic by observing cell viability. After evaluating these models, the best-performing model is integrated into the web-based application Toxicheck. Toxicheck is an in-silico tool for predicting the toxicological profile of a nanomaterial based on the selected features that were used to train the GA-RF with the SMOTE model. With its simple and intuitive user interface, it is easy to navigate and operate.

Considering the time it takes for traditional methods to test the toxicity of nanomaterials through in-vitro and in-vivo methods, this application offers a faster way to test nanomaterial samples through machine learning. By simulating how cell assays are affected by a nanoparticle as described by its physicochemical properties, this approach is valuable for researchers and ENM manufacturers as it provides insights into the toxicity of the nanomaterial that they will be handling.

During the development of the models, multiple data-handling techniques are applied to improve data quality. The dataset has several String-type variables. For the dataset, LabelEncoder() is used to convert binary categorical variables into numerical data, while the multiple categorical variables are dropped due to a lack of experience in handling one-hot encoding results on the researcher's part. Upon inspecting each model's results, the repudiation of these variables may have cost a significant amount of scores to the metrics and may have reduced the dataset's quality instead.

Feature scaling is applied to the predictors specifically to scale the values for diameter, concentration, and exposure time, which all have a wide range of numerical data. However, the use of standard scaler worsens the metric scores to an unacceptable state. Most of the performance metrics drop below 0.50, with score reduction reaching as high as 65%. Lastly, SMOTE is applied to address the imbalance between the toxic and non-toxic classes.

The use of hybrid algorithms for building nano-QSAR models proves to be effective, as shown by genetic algorithm's positive influence on the performance metrics of the base classifier algorithms. GA significantly improved every model's accuracy by 9% to 19%. However, it has mixed effects on the other metrics, and it significantly reduced AUC scores by 8% to 10%.

GA-RF provided the best metrics relative to all the models tested in this study. Its scores were improved (at most 1%) by applying SMOTE to address the class imbalance of the dataset. However, when compared to the metrics from related literature, the scores produced by the models in this study are relatively lower. This could be attributed to the dataset's quality and the functions used in model training.

Note that the majority of these models still hold merit, with their metric scores reaching 80% on average, and they can still be used as a reliable way to look for valuable insights needed for nanomaterial-related decision-making. Integrating the best-performing model into a web application gives professionals and researchers an intuitive way to assess a nanomaterial safely using the in-silico approach. The system can be used alongside traditional toxicity testing approaches for cross-validating results. Overall, this study emphasizes the significance of nanotoxicity testing and offers new directions for future research into mitigating the harmful effects of NPs and ENMs by investigating and confirming the applicability of hybrid algorithms in this field.

VII. Conclusions

This research showcases the potential of in-silico toxicity testing in classifying the toxicological profile of a nanomaterial when combined with hybrid algorithm. After obtaining the dataset from [8], the researcher applies exploratory data analysis and data preprocessing using label encoding and correlation coefficients. The processed data is passed to the feature selection process, where genetic algorithm is involved. Genetic algorithm is run on each base classifier to determine which features are selected for each distinct model. Unselected predictors are dropped before proceeding to data splitting.

The dataset is divided into 80:20, where 80% is for the training set and 20% is for the testing set. The training set is duplicated into four copies: the first one has no techniques applied, the second has standard scaler applied, the third has SMOTE applied, and the last has both standard scaler and SMOTE applied. These four copies are also done for the models with GA applied. Overall, there are 8 total versions of the training set for each base classifier algorithm: the first 4 are for the original dataset, and the remaining is for the dataset with GA. The original dataset is still utilized to run the individual base classifier models without the effect of feature selection to serve as the control setup of the study.

Afterward, hyperparameter tuning is applied to all prepared models. The resulting models are trained and evaluated using the performance metrics: accuracy, precision, recall, F1, ROC-AUC, and MCC, where MCC is the main metric to determine the best-performing model. Results show that GA-RF with SMOTE is the optimal model among the selections and is integrated into the web application named Toxicheck. This study confirms the applicability and efficiency of hybrid algorithms for classifying nanomaterial toxicity using a diverse dataset. This enables the users of the system to perform safer and faster testing on a wider selection of nanomaterials using an in-silico approach.

VIII. Recommendations

In terms of improving the performance of the models using the same dataset, future studies may attempt to investigate the effect of removing outliers from the dataset. It is also recommended to try applying one-hot encoding to the multiple categorical features that were previously dropped in this study, as repudiating these variables may have reduced the dataset's quality.

Additionally, investigations into other feature scaling, oversampling, and feature selection techniques will be helpful in establishing a more solid foundation regarding the effect of these techniques on the model. Future researchers may also develop other classification models using other algorithms that were not tested in this study (such as XGBoost, Adaboost, etc.) with GA as a feature selector.

Alternatively, if the researcher uses a different dataset, they can still use the same machine learning algorithms in the study to expand its scope. Note that it is recommended to find a balanced dataset that still covers a large volume of different nanoparticles to maintain a wide applicability domain. Lastly, future studies can also focus on testing other toxicity measures besides cell viability.

In terms of improving the web application, a feature that lets the user choose what model they want to use for the prediction can be added, assuming that all other models in this study are also integrated into the system. Alternatively, the results and predictions of all the models can be shown side by side in the web application to give the user an insight into how each model predicted the classification of the nanomaterial they described.

IX. Bibliography

[1] E. R. Bandala and M. Berli, "Engineered nanomaterials (enms) and their role at the nexus of food, energy, and water," *Materials Science for Energy Technologies*, vol. 2, pp. 29–40, 2019.

[2] S. Balraadjsing, W. J. G. M. Peijnenburg, and M. G. Vijver, "Exploring the potential of in silico machine learning tools for the prediction of acute daphnia magna nanotoxicity," *Chemosphere*, vol. 307, 2022.

[3] G. Gul, R. Yildirim, and N. Ileri-Ercan, "Cytotoxicity analysis of nanoparticles by association rule mining," *Environmental Science: Nano*, vol. 8, p. 937–949, 2021.

[4] E. Kabir, V. Kumar, K.-H. Kim, A. C. K. Yip, and J. R. Sohn, "Environmental impacts of nanomaterials," *Journal of Environmental Management*, vol. 225, pp. 261–271, 2018.

[5] I. Furxhi, F. Murphy, M. Mullins, and C. A. Poland, "Machine learning prediction of nanoparticle in vitro toxicity: A comparative study of classifiers and ensemble-classifiers using the copeland index," *Toxicology Letters*, vol. 312, pp. 157–166, 2019.

[6] V. Forest, J.-F. Hochepied, L. Leclerc, A. Trouvé, K. Abdelkebir, G. Sarry, V. Augusto, and J. Pourchez, "Towards an alternative to nano-qsar for nanoparticle toxicity ranking in case of small datasets," *Journal of Nanoparticle Research*, vol. 21, 2019.

[7] R. Divya and R. Shantha Selva Kumari, "Genetic algorithm with logistic regression feature selection for alzheimer's disease classification," *Neural Computing and Applications*, vol. 33, pp. 8435–8444, 2021.

[8] H. I. Labouta, N. Asgarian, K. Rinker, and D. T. Cramb, "Meta-analysis of nanoparticle cytotoxicity via data-mining the literature," *ACS Nano*, 2019.

[9] H. M. Ahmed, A. Roy, M. Wahab, M. Ahmed, G. Othman-Qadir, B. H. Elesawy, M. U. Khandaker, M. N. Islam, and T. B. Emran, "Applications of nanomaterials in agrifood and pharmaceutical industry," *Journal of Nanomaterials*, vol. 2021, pp. 1–10, 2021.

[10] M. Martinez, "Differences between in vitro, in vivo and in silico assays in preclinical research," *ZeClinics*, 2022.

[11] E. Fröhlich, "Comparison of conventional and advanced in vitro models in the toxicity testing of nanoparticles," *Artificial Cells, Nanomedicine, and Biotechnology*, vol. 46, p. 1091–1107, 2018.

[12] T. X. Trinh, M. Seo, T. H. Yoon, and J. Kim, "Developing random forest based qsar models for predicting the mixture toxicity of tio2 based nano-mixtures to daphnia magna," *NanoImpact*, vol. 25, 2022.

[13] B.-H. Mao, Y.-K. Luo, B.-J. Wang, C.-W. Chen, F.-Y. Cheng, Y.-H. Lee, S.-J. Yan, and Y.-J. Wang, "Use of an in-silico knowledge discovery approach to determine mechanistic studies of silver nanoparticles-induced toxicity from in vitro to in vivo," *Particle and Fibre Toxicology*, vol. 19, 2022.

[14] J. Cao, Y. Pan, Y. Jiang, R. Qi, B. Yuan, Z. Jia, J. Jiang, and Q. Wang, "Computer-aided nanotoxicology: risk assessment of metal oxide nanoparticles via nano-qsar," *Green Chemistry*, vol. 22, pp. 3512–3521, 2020.

[15] M. K. Ha, T. X. Trinh, J. S. Choi, D. Maulina, H. G. Byun, and T. H. Yoon, "Toxicity classification of oxide nanomaterials: Effects of data gap filling and pchem score-based screening approaches," *Scientific Reports*, vol. 8, 2018.

[16] D. van der Merwe and J. A. Pickrell, *Chapter 18 - Toxicity of Nanomaterials*, pp. 319–326. Academic Press, third edition ed., 2018.

[17] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *arXiv.org*, Jun 2011.

[18] M. Abdelrahim, C. Merlosy, and T. Wang, "Hybrid machine learning approaches: A method to improve expected output of semi-structured sequential data," in *2016 IEEE Tenth International Conference on Semantic Computing (ICSC)*, IEEE, Feb 2016.

[19] C. M. Boukhater, O. Dakroub, F. Lahoud, M. Awad, and H. Artail, "An intelligent and fair GA carpooling scheduler as a social solution for greener transportation," in *MELECON 2014 - 2014 17th IEEE Mediterranean Electrotechnical Conference*, IEEE, Apr. 2014.

[20] M. Banoula, "An introduction to logistic regression in python," Sep 2022.

[21] Y.-S. Park and S. Lek, *Chapter 7 - Artificial Neural Networks: Multilayer Perceptron for Ecological Modeling*, vol. 28 of *Developments in Environmental Modelling*, pp. 123–140. Elsevier, 2016.

[22] Simplilearn, "Random forest algorithm," Jan 2023.

X. Appendix

A. Source Code

1. **Machine Learning: GA-LR.py**

```python
# ———————————————————— IMPORTANT ———————————————————— #
# This is a cleaned version of the ipynb files that was used to develop the models.
# It is only formatted in python SOLELY for CLEANER presentation in the SP paper.
# ——————————————————————— END ——————————————————————— #

# Importing Libraries
import io
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import pickle
import seaborn as sns
import warnings
from genetic_selection import GeneticSelectionCV
from imblearn.over_sampling import SMOTE
from sklearn.metrics import accuracy_score, precision_score, f1_score, recall_score
from sklearn.metrics import confusion_matrix, roc_auc_score, roc_curve, matthews_corrcoef
from sklearn.metrics import classification_report
from sklearn.model_selection import GridSearchCV, train_test_split, StratifiedKFold
from sklearn.preprocessing import LabelEncoder, MultiLabelBinarizer, StandardScaler
from sklearn.linear_model import LogisticRegression

# Upload Data
dataset = pd.read_csv(r'np.csv')
dataset = dataset.dropna()
print(dataset)

# Show Count of Unique Values per Column
dataset['Cell Type'].nunique()

# Show Unique Values per Column
dataset['Cell Type'].unique()
print(dataset.dtypes)

# Splitting Data into Predictors and Response
predictors = dataset.iloc[:, 0:9]
response = dataset.iloc[:, 9]

# Check for Imbalance
dataset_eda = dataset.copy()
dataset_eda['classification'].value_counts()

# Feature Selection
model = LogisticRegression(random_state=2,max_iter=1000)
cv = StratifiedKFold(n_splits=5, shuffle=True)
GALR = GeneticSelectionCV(
            model,
            cv=cv,
            verbose=0,
            scoring="accuracy",
            max_features=9,
            n_population=100,
            crossover_proba=0.5,
            mutation_proba=0.2,
            n_generations=50,
            crossover_independent_proba=0.5,
            mutation_independent_proba=0.04,
            tournament_size=3,
            n_gen_no_change=10,
            caching=True,
            n_jobs=-1)
GALR = GALR.fit(predictors, response)
print('Features:', predictors.columns[GALR.support_])

# Drop Non-Selected Features
predictors_ga = predictors.drop(['Colloidal Stability Checked', 'Interference Checked'],
                                axis = 1)
print(predictors_ga)

# Data Splitting
seed = 2
test_size = 0.20
X_train, X_test, y_train, y_test = train_test_split(predictors, response,
                                                    test_size=test_size,
                                                    random_state=seed)

X_train_ga, X_test_ga, y_train_ga, y_test_ga = train_test_split(predictors_ga, response,
                                                                test_size=test_size,
                                                                                random_state=seed)

# Check Number of Rows
X_train.shape[0], X_test.shape[0], X_train_ga.shape[0], X_test_ga.shape[0]

# Feature Scaling
scaler = StandardScaler()
X_train_std = scaler.fit_transform(X_train)
X_train_scaled = pd.DataFrame(X_train_std, index=X_train.index, columns=X_train.columns)
print(X_train_scaled)

# Feature Scaling for GA Model
X_train_gastd = scaler.fit_transform(X_train_ga)
X_train_gascaled = pd.DataFrame(X_train_gastd, index=X_train_ga.index,
                                columns=X_train_ga.columns)
print(X_train_gascaled)

# Class Balancing
sm = SMOTE(random_state=2)
X_train_sm, y_train_sm = sm.fit_resample(X_train, y_train)
X_train_sm_scaled, y_train_sm_scaled = sm.fit_resample(X_train_scaled, y_train)
X_train_all, y_train_all = sm.fit_resample(X_train_gascaled, y_train_ga)

# ---------------- Logistic Regression (No Feature Scaling) ------------- #
model = LogisticRegression(random_state=2, max_iter=1000)
warnings.filterwarnings('ignore')

# Hyperparameter Tuning
parameters = {
    'penalty' : ['l1','l2'],
    'C'       : np.logspace(-3,3,7),
    'solver'  : ['newton-cg', 'lbfgs', 'liblinear'],
}

# For Model 1
cv = StratifiedKFold(n_splits=5, shuffle=True)
model1 = GridSearchCV(model, parameters, cv=cv)
model1.fit(X_train, y_train)
print(model1.best_params_)
model1.best_estimator_.score(X_test, y_test)

# For Model 1 GA
model1_ga = GridSearchCV(model, parameters, cv=cv)
model1_ga.fit(X_train_ga, y_train_ga)
print(model1_ga.best_params_)
model1_ga.best_estimator_.score(X_test_ga, y_test_ga)

# Make Predictions for Test Data
y_pred = model1.predict(X_test)
y_pred_ga = model1_ga.predict(X_test_ga)

# Evaluate Predictions for Model 1
mcc = matthews_corrcoef(y_test_ga,y_pred_ga)
confusion_mat = confusion_matrix(y_test_ga,y_pred_ga)
print("MCC is:",mcc)
print("Confusion Matrix")
print(confusion_mat)
print(classification_report(y_test_ga,y_pred_ga))

# Evaluate Predictions for Model 1 GA
mcc = matthews_corrcoef(y_test_ga,y_pred_ga)
confusion_mat = confusion_matrix(y_test_ga,y_pred_ga)
print("MCC is:",mcc)
print("Confusion Matrix")
print(confusion_mat)
print(classification_report(y_test_ga,y_pred_ga))

# --- For Model 1 ---
fpr, tpr, threshold = roc_curve(y_test, y_pred, pos_label=1)
random = [0 for i in range(len(y_test))]
p_fpr, p_tpr, _ = roc_curve(y_test, random, pos_label=1)
auc_score = roc_auc_score(y_test, y_pred)*100

print("AUC Score: ", auc_score)

# Plot ROC Curves
plt.plot(fpr, tpr, linestyle='--',color='blue', label='Logistic Regression')
plt.plot(p_fpr, p_tpr, linestyle='--', color='black')

# Title and Labels
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')

#Legend
plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show()

# --- For Model 1 GA ---
fpr, tpr, threshold = roc_curve(y_test_ga, y_pred_ga, pos_label=1)
random = [0 for i in range(len(y_test_ga))]
```

```
p_fpr, p_tpr, _ = roc_curve(y_test_ga, random, pos_label=1)
auc_score = roc_auc_score(y_test_ga, y_pred_ga)*100
print("AUC Score: ", auc_score)

# Plot ROC Curves
plt.plot(fpr, tpr, linestyle='--',color='blue', label='Logistic Regression')
plt.plot(p_fpr, p_tpr, linestyle='--', color='black')

# Title and Labels
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')

#Legend
plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show()

# ---------------- Logistic Regression (With Feature Scaling) ------------- #
#For Model 2
model2 = GridSearchCV(model, parameters, cv=cv)
model2.fit(X_train_scaled, y_train)
print(model2.best_params_)
model2.best_estimator_.score(X_test, y_test)

# For Model 2 GA
model2_ga = GridSearchCV(model, parameters, cv=cv)
model2_ga.fit(X_train_gascaled, y_train_ga)
print(model2_ga.best_params_)
model2_ga.best_estimator_.score(X_test_ga, y_test_ga)

# Make Predictions for Test Data
y_pred = model2.predict(X_test)
y_pred_ga = model2_ga.predict(X_test_ga)

# Evaluate Predictions for Model 2
mcc = matthews_corrcoef(y_test_ga,y_pred_ga)
confusion_mat = confusion_matrix(y_test_ga,y_pred_ga)
print("MCC is:",mcc)
print("Confusion Matrix")
print(confusion_mat)
print(classification_report(y_test_ga,y_pred_ga))

# Evaluate Predictions for Model 2 GA
mcc = matthews_corrcoef(y_test_ga,y_pred_ga)
confusion_mat = confusion_matrix(y_test_ga,y_pred_ga)
print("MCC is:",mcc)
print("Confusion Matrix")
print(confusion_mat)
print(classification_report(y_test_ga,y_pred_ga))

# --- For Model 2 ---
fpr, tpr, threshold = roc_curve(y_test, y_pred, pos_label=1)
random = [0 for i in range(len(y_test))]
p_fpr, p_tpr, _ = roc_curve(y_test, random, pos_label=1)
auc_score = roc_auc_score(y_test, y_pred)*100

print("AUC Score: ", auc_score)

# Plot ROC Curves
plt.plot(fpr, tpr, linestyle='--',color='blue', label='Logistic Regression')
plt.plot(p_fpr, p_tpr, linestyle='--', color='black')

# Title and Labels
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')

#Legend
plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show()

# --- For Model 2 GA ---
fpr, tpr, threshold = roc_curve(y_test_ga, y_pred_ga, pos_label=1)
random = [0 for i in range(len(y_test_ga))]
p_fpr, p_tpr, _ = roc_curve(y_test_ga, random, pos_label=1)
auc_score = roc_auc_score(y_test_ga, y_pred_ga)*100
print("AUC Score: ", auc_score)

# Plot ROC Curves
plt.plot(fpr, tpr, linestyle='--',color='blue', label='Logistic Regression')
plt.plot(p_fpr, p_tpr, linestyle='--', color='black')

# Title and Labels
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')

#Legend
plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show()

# ––––––––––––– Logistic Regression (With Smote) ––––––––––– #
# For Model 3
model3 = GridSearchCV(model, parameters, cv=cv)
model3.fit(X_train_sm, y_train_sm)
print(model3.best_params_)
model3.best_estimator_.score(X_test, y_test)

# For Model 3 GA
model3_ga = GridSearchCV(model, parameters, cv=cv)
model3_ga.fit(X_train_ga, y_train_ga)
print(model3_ga.best_params_)
model3_ga.best_estimator_.score(X_test_ga, y_test_ga)

# Make Predictions for Test Data
y_pred = model3.predict(X_test)
y_pred_ga = model3_ga.predict(X_test_ga)

# Evaluate Predictions for Model 3
mcc = matthews_corrcoef(y_test, y_pred)
confusion_mat = confusion_matrix(y_test,y_pred)
print("MCC is:",mcc)
print("Confusion Matrix")
print(confusion_mat)
print(classification_report(y_test,y_pred))

# Evaluate Predictions for Model 3 GA
mcc = matthews_corrcoef(y_test_ga,y_pred_ga)
confusion_mat = confusion_matrix(y_test_ga,y_pred_ga)
print("MCC is:",mcc)
print("Confusion Matrix")
print(confusion_mat)
print(classification_report(y_test_ga,y_pred_ga))

# ––– For Model 3 –––
fpr, tpr, threshold = roc_curve(y_test, y_pred, pos_label=1)
random = [0 for i in range(len(y_test))]
p_fpr, p_tpr, _ = roc_curve(y_test, random, pos_label=1)
auc_score = roc_auc_score(y_test, y_pred)*100

print("AUC Score: ", auc_score)

# Plot ROC Curves
plt.plot(fpr, tpr, linestyle='--',color='blue', label='Logistic Regression')
plt.plot(p_fpr, p_tpr, linestyle='--', color='black')

# Title and Labels
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')

#Legend
plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show()

# ––– For Model 3 GA –––
fpr, tpr, threshold = roc_curve(y_test_ga, y_pred_ga, pos_label=1)
random = [0 for i in range(len(y_test_ga))]
p_fpr, p_tpr, _ = roc_curve(y_test_ga, random, pos_label=1)
auc_score = roc_auc_score(y_test_ga, y_pred_ga)*100
print("AUC Score: ", auc_score)

# Plot ROC Curves
plt.plot(fpr, tpr, linestyle='--',color='blue', label='Logistic Regression')
plt.plot(p_fpr, p_tpr, linestyle='--', color='black')

# Title and Labels
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')

#Legend
plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show()

# ––––––––––––– Logistic Regression (All Applied) ––––––––––– #
# For Model 4
model4 = GridSearchCV(model, parameters, cv=cv)
model4.fit(X_train_sm_scaled, y_train_sm_scaled)
print(model4.best_params_)
model4.best_estimator_.score(X_test, y_test)

# For Model 4 GA
model4_ga = GridSearchCV(model, parameters, cv=cv)
model4_ga.fit(X_train_all, y_train_all)
print(model4_ga.best_params_)
model4_ga.best_estimator_.score(X_test_ga, y_test_ga)

# Make Predictions for Test Data
```

```
y_pred = model4.predict(X_test)
y_pred_ga = model4_ga.predict(X_test_ga)

# Evaluate Predictions for Model 4
mcc = matthews_corrcoef(y_test, y_pred)
confusion_mat = confusion_matrix(y_test,y_pred)
print("MCC is:",mcc)
print("Confusion Matrix")
print(confusion_mat)
print(classification_report(y_test,y_pred))

# Evaluate Predictions for Model 4 GA
mcc = matthews_corrcoef(y_test_ga,y_pred_ga)
confusion_mat = confusion_matrix(y_test_ga,y_pred_ga)
print("MCC is:",mcc)
print("Confusion Matrix")
print(confusion_mat)
print(classification_report(y_test_ga,y_pred_ga))

# --- For Model 4 ---
fpr, tpr, threshold = roc_curve(y_test, y_pred, pos_label=1)
random = [0 for i in range(len(y_test))]
p_fpr, p_tpr, _ = roc_curve(y_test, random, pos_label=1)
auc_score = roc_auc_score(y_test, y_pred)*100
print("AUC Score: ", auc_score)

# Plot ROC Curves
plt.plot(fpr, tpr, linestyle='--',color='blue', label='Logistic Regression')
plt.plot(p_fpr, p_tpr, linestyle='--', color='black')

# Title and Labels
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')

#Legend
plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show()

# --- For Model 4 GA ---
fpr, tpr, threshold = roc_curve(y_test_ga, y_pred_ga, pos_label=1)
random = [0 for i in range(len(y_test_ga))]
p_fpr, p_tpr, _ = roc_curve(y_test_ga, random, pos_label=1)
auc_score = roc_auc_score(y_test_ga, y_pred_ga)*100
print("AUC Score: ", auc_score)

# Plot ROC Curves
plt.plot(fpr, tpr, linestyle='--',color='blue', label='Logistic Regression')
plt.plot(p_fpr, p_tpr, linestyle='--', color='black')

# Title and Labels
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')

#Legend
plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show()

# ------------- Saving the Pipeline into a File ------------ #
pickle.dump(model3_ga, open('galr.pkl', 'wb'))
```

2. **Machine Learning: GA-ANN.py**

```
# ------------------------------- IMPORTANT ------------------------------- #
# This is a cleaned version of the ipynb files that was used to develop the models.
# It is only formatted in python SOLELY for CLEANER presentation in the SP paper.
# ---------------------------------- END ---------------------------------- #

# Importing Libraries
import io
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import pickle
import seaborn as sns
import warnings
from genetic_selection import GeneticSelectionCV
from imblearn.over_sampling import SMOTE
from sklearn.metrics import accuracy_score, precision_score, f1_score, recall_score
from sklearn.metrics import confusion_matrix, roc_auc_score, roc_curve, matthews_corrcoef
from sklearn.metrics import classification_report
from sklearn.model_selection import GridSearchCV, train_test_split, StratifiedKFold
from sklearn.preprocessing import LabelEncoder, MultiLabelBinarizer, StandardScaler
from sklearn.neural_network import MLPClassifier

# Upload Data
```

```
dataset = pd.read_csv(r'np.csv')
dataset = dataset.dropna()
print(dataset)

# Show Count of Unique Values per Column
dataset['Cell Type'].nunique()

# Show Unique Values per Column
dataset['Cell Type'].unique()
print(dataset.dtypes)

# Splitting Data into Predictors and Response
predictors = dataset.iloc[:, 0:9]
response = dataset.iloc[:, 9]

# Check for Imbalance
dataset_eda = dataset.copy()
dataset_eda['classification'].value_counts()

# Feature Selection
model = MLPClassifier(random_state=2, max_iter=1000)
cv = StratifiedKFold(n_splits=5, shuffle=True)
GAANN = GeneticSelectionCV(
            model,
            cv=cv,
            verbose=0,
            scoring="accuracy",
            max_features=9,
            n_population=100,
            crossover_proba=0.5,
            mutation_proba=0.2,
            n_generations=50,
            crossover_independent_proba=0.5,
            mutation_independent_proba=0.04,
            tournament_size=3,
            n_gen_no_change=10,
            caching=True,
            n_jobs=-1)
GAANN = GAANN.fit(predictors, response)
print('Features:', predictors.columns[GAANN.support_])

# Drop Non-Selected Features
predictors_ga = predictors.drop(['NP Type', 'Cell Culture', 'Interference Checked'],
                                axis = 1)
print(predictors_ga)

# Data Splitting
seed = 2
test_size = 0.20
X_train, X_test, y_train, y_test = train_test_split(predictors, response,
                                                    test_size=test_size,
                                                    random_state=seed)

X_train_ga, X_test_ga, y_train_ga, y_test_ga = train_test_split(predictors_ga, response,
                                                                test_size=test_size,
                                                                random_state=seed)

# Check Number of Rows
X_train.shape[0], X_test.shape[0], X_train_ga.shape[0], X_test_ga.shape[0]

# Feature Scaling
scaler = StandardScaler()
X_train_std = scaler.fit_transform(X_train)
X_train_scaled = pd.DataFrame(X_train_std, index=X_train.index, columns=X_train.columns)
print(X_train_scaled)

# Feature Scaling for GA Model
X_train_gastd = scaler.fit_transform(X_train_ga)
X_train_gascaled = pd.DataFrame(X_train_gastd, index=X_train_ga.index,
                                columns=X_train_ga.columns)
print(X_train_gascaled)

# Class Balancing
sm = SMOTE(random_state=2)
X_train_sm, y_train_sm = sm.fit_resample(X_train, y_train)
X_train_sm_scaled, y_train_sm_scaled = sm.fit_resample(X_train_scaled, y_train)
X_train_all, y_train_all = sm.fit_resample(X_train_gascaled, y_train_ga)

# ---------------- Artificial Neural Network (No Feature Scaling) ------------- #
model = MLPClassifier(random_state=2, max_iter=1000)
warnings.filterwarnings('ignore')

# Hyperparameter Tuning
parameters = {
    'hidden_layer_sizes': [(10,30,10),(20,),(100,)],
    'alpha': [0.01, 0.05],
    'learning_rate_init': [0.001, 0.01, 0.1],
}

# For Model 1
cv = StratifiedKFold(n_splits=5, shuffle=True)
model1 = GridSearchCV(model, parameters, cv=cv)
```

```
model1.fit(X_train, y_train)
print(model1.best_params_)
model1.best_estimator_.score(X_test, y_test)

# For Model 1 GA
model1_ga = GridSearchCV(model, parameters, cv=cv)
model1_ga.fit(X_train_ga, y_train_ga)
print(model1_ga.best_params_)
model1_ga.best_estimator_.score(X_test_ga, y_test_ga)

# Make Predictions for Test Data
y_pred = model1.predict(X_test)
y_pred_ga = model1_ga.predict(X_test_ga)

# Evaluate Predictions for Model 1
mcc = matthews_corrcoef(y_test_ga,y_pred_ga)
confusion_mat = confusion_matrix(y_test_ga,y_pred_ga)
print("MCC is:",mcc)
print("Confusion Matrix")
print(confusion_mat)
print(classification_report(y_test_ga,y_pred_ga))

# Evaluate Predictions for Model 1 GA
mcc = matthews_corrcoef(y_test_ga,y_pred_ga)
confusion_mat = confusion_matrix(y_test_ga,y_pred_ga)
print("MCC is:",mcc)
print("Confusion Matrix")
print(confusion_mat)
print(classification_report(y_test_ga,y_pred_ga))

# --- For Model 1 ---
fpr, tpr, threshold = roc_curve(y_test, y_pred, pos_label=1)
random = [0 for i in range(len(y_test))]
p_fpr, p_tpr, _ = roc_curve(y_test, random, pos_label=1)
auc_score = roc_auc_score(y_test, y_pred)*100

print("AUC Score: ", auc_score)

# Plot ROC Curves
plt.plot(fpr, tpr, linestyle='--',color='blue', label='Artificial Neural Network')
plt.plot(p_fpr, p_tpr, linestyle='--', color='black')

# Title and Labels
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')

#Legend
plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show()

# --- For Model 1 GA ---
fpr, tpr, threshold = roc_curve(y_test_ga, y_pred_ga, pos_label=1)
random = [0 for i in range(len(y_test_ga))]
p_fpr, p_tpr, _ = roc_curve(y_test_ga, random, pos_label=1)
auc_score = roc_auc_score(y_test_ga, y_pred_ga)*100
print("AUC Score: ", auc_score)

# Plot ROC Curves
plt.plot(fpr, tpr, linestyle='--',color='blue', label='Artificial Neural Network')
plt.plot(p_fpr, p_tpr, linestyle='--', color='black')

# Title and Labels
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')

#Legend
plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show()

# ----------------- Artificial Neural Network (With Feature Scaling) ----------------- #
#For Model 2
model2 = GridSearchCV(model, parameters, cv=cv)
model2.fit(X_train_scaled, y_train)
print(model2.best_params_)
model2.best_estimator_.score(X_test, y_test)

# For Model 2 GA
model2_ga = GridSearchCV(model, parameters, cv=cv)
model2_ga.fit(X_train_gascaled, y_train_ga)
print(model2_ga.best_params_)
model2_ga.best_estimator_.score(X_test_ga, y_test_ga)

# Make Predictions for Test Data
y_pred = model2.predict(X_test)
y_pred_ga = model2_ga.predict(X_test_ga)

# Evaluate Predictions for Model 2
mcc = matthews_corrcoef(y_test_ga,y_pred_ga)
```

```
confusion_mat = confusion_matrix(y_test_ga,y_pred_ga)
print("MCC is:",mcc)
print("Confusion Matrix")
print(confusion_mat)
print(classification_report(y_test_ga,y_pred_ga))

# Evaluate Predictions for Model 2 GA
mcc = matthews_corrcoef(y_test_ga,y_pred_ga)
confusion_mat = confusion_matrix(y_test_ga,y_pred_ga)
print("MCC is:",mcc)
print("Confusion Matrix")
print(confusion_mat)
print(classification_report(y_test_ga,y_pred_ga))

# --- For Model 2 ---
fpr, tpr, threshold = roc_curve(y_test, y_pred, pos_label=1)
random = [0 for i in range(len(y_test))]
p_fpr, p_tpr, _ = roc_curve(y_test, random, pos_label=1)
auc_score = roc_auc_score(y_test, y_pred)*100

print("AUC Score: ", auc_score)

# Plot ROC Curves
plt.plot(fpr, tpr, linestyle='--',color='blue', label='Artificial Neural Network')
plt.plot(p_fpr, p_tpr, linestyle='--', color='black')

# Title and Labels
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')

#Legend
plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show()

# --- For Model 2 GA ---
fpr, tpr, threshold = roc_curve(y_test_ga, y_pred_ga, pos_label=1)
random = [0 for i in range(len(y_test_ga))]
p_fpr, p_tpr, _ = roc_curve(y_test_ga, random, pos_label=1)
auc_score = roc_auc_score(y_test_ga, y_pred_ga)*100
print("AUC Score: ", auc_score)

# Plot ROC Curves
plt.plot(fpr, tpr, linestyle='--',color='blue', label='Artificial Neural Network')
plt.plot(p_fpr, p_tpr, linestyle='--', color='black')

# Title and Labels
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')

#Legend
plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show()

# ---------------- Artificial Neural Network (With Smote) -------------- #
# For Model 3
model3 = GridSearchCV(model, parameters, cv=cv)
model3.fit(X_train_sm, y_train_sm)
print(model3.best_params_)
model3.best_estimator_.score(X_test, y_test)

# For Model 3 GA
model3_ga = GridSearchCV(model, parameters, cv=cv)
model3_ga.fit(X_train_ga, y_train_ga)
print(model3_ga.best_params_)
model3_ga.best_estimator_.score(X_test_ga, y_test_ga)

# Make Predictions for Test Data
y_pred = model3.predict(X_test)
y_pred_ga = model3_ga.predict(X_test_ga)

# Evaluate Predictions for Model 3
mcc = matthews_corrcoef(y_test, y_pred)
confusion_mat = confusion_matrix(y_test,y_pred)
print("MCC is:",mcc)
print("Confusion Matrix")
print(confusion_mat)
print(classification_report(y_test,y_pred))

# Evaluate Predictions for Model 3 GA
mcc = matthews_corrcoef(y_test_ga,y_pred_ga)
confusion_mat = confusion_matrix(y_test_ga,y_pred_ga)
print("MCC is:",mcc)
print("Confusion Matrix")
print(confusion_mat)
print(classification_report(y_test_ga,y_pred_ga))

# --- For Model 3 ---
fpr, tpr, threshold = roc_curve(y_test, y_pred, pos_label=1)
```

```
random = [0 for i in range(len(y_test))]
p_fpr, p_tpr, _ = roc_curve(y_test, random, pos_label=1)
auc_score = roc_auc_score(y_test, y_pred)*100

print("AUC Score: ", auc_score)

# Plot ROC Curves
plt.plot(fpr, tpr, linestyle='--',color='blue', label='Artificial Neural Network')
plt.plot(p_fpr, p_tpr, linestyle='--', color='black')

# Title and Labels
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')

#Legend
plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show()

# --- For Model 3 GA ---
fpr, tpr, threshold = roc_curve(y_test_ga, y_pred_ga, pos_label=1)
random = [0 for i in range(len(y_test_ga))]
p_fpr, p_tpr, _ = roc_curve(y_test_ga, random, pos_label=1)
auc_score = roc_auc_score(y_test_ga, y_pred_ga)*100
print("AUC Score: ", auc_score)

# Plot ROC Curves
plt.plot(fpr, tpr, linestyle='--',color='blue', label='Artificial Neural Network')
plt.plot(p_fpr, p_tpr, linestyle='--', color='black')

# Title and Labels
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')

#Legend
plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show()

# ---------------- Artificial Neural Network (All Applied) ------------- #
# For Model 4
model4 = GridSearchCV(model, parameters, cv=cv)
model4.fit(X_train_sm_scaled, y_train_sm_scaled)
print(model4.best_params_)
model4.best_estimator_.score(X_test, y_test)

# For Model 4 GA
model4_ga = GridSearchCV(model, parameters, cv=cv)
model4_ga.fit(X_train_all, y_train_all)
print(model4_ga.best_params_)
model4_ga.best_estimator_.score(X_test_ga, y_test_ga)

# Make Predictions for Test Data
y_pred = model4.predict(X_test)
y_pred_ga = model4_ga.predict(X_test_ga)

# Evaluate Predictions for Model 4
mcc = matthews_corrcoef(y_test, y_pred)
confusion_mat = confusion_matrix(y_test,y_pred)
print("MCC is:",mcc)
print("Confusion Matrix")
print(confusion_mat)
print(classification_report(y_test,y_pred))

# Evaluate Predictions for Model 4 GA
mcc = matthews_corrcoef(y_test_ga,y_pred_ga)
confusion_mat = confusion_matrix(y_test_ga,y_pred_ga)
print("MCC is:",mcc)
print("Confusion Matrix")
print(confusion_mat)
print(classification_report(y_test_ga,y_pred_ga))

# --- For Model 4 ---
fpr, tpr, threshold = roc_curve(y_test, y_pred, pos_label=1)
random = [0 for i in range(len(y_test))]
p_fpr, p_tpr, _ = roc_curve(y_test, random, pos_label=1)
auc_score = roc_auc_score(y_test, y_pred)*100
print("AUC Score: ", auc_score)

# Plot ROC Curves
plt.plot(fpr, tpr, linestyle='--',color='blue', label='Artificial Neural Network')
plt.plot(p_fpr, p_tpr, linestyle='--', color='black')

# Title and Labels
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')

#Legend
plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show()

# —— For Model 4 GA ——
fpr, tpr, threshold = roc_curve(y_test_ga, y_pred_ga, pos_label=1)
random = [0 for i in range(len(y_test_ga))]
p_fpr, p_tpr, _ = roc_curve(y_test_ga, random, pos_label=1)
auc_score = roc_auc_score(y_test_ga, y_pred_ga)*100
print("AUC Score: ", auc_score)

# Plot ROC Curves
plt.plot(fpr, tpr, linestyle='--',color='blue', label='Artificial Neural Network')
plt.plot(p_fpr, p_tpr, linestyle='--', color='black')

# Title and Labels
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')

#Legend
plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show()

# ———————— Saving the Pipeline into a File —————— #
pickle.dump(model3_ga, open('gaann.pkl', 'wb'))
```

3. **Machine Learning: GA-RF.py**

```
# ——————————————————— IMPORTANT ——————————————————— #
# This is a cleaned version of the ipynb files that was used to develop the models.
# It is only formatted in python SOLELY for CLEANER presentation in the SP paper.
# ——————————————————— END ——————————————————— #

# Importing Libraries
import io
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import pickle
import seaborn as sns
import warnings
from genetic_selection import GeneticSelectionCV
from imblearn.over_sampling import SMOTE
from sklearn.metrics import accuracy_score, precision_score, f1_score, recall_score
from sklearn.metrics import confusion_matrix, roc_auc_score, roc_curve, matthews_corrcoef
from sklearn.metrics import classification_report
from sklearn.model_selection import GridSearchCV, train_test_split, StratifiedKFold
from sklearn.preprocessing import LabelEncoder, MultiLabelBinarizer, StandardScaler
from sklearn.ensemble import RandomForestClassifier

# Upload Data
dataset = pd.read_csv(r'np.csv')
dataset = dataset.dropna()
print(dataset)

# Show Count of Unique Values per Column
dataset['Cell Type'].nunique()

# Show Unique Values per Column
dataset['Cell Type'].unique()
print(dataset.dtypes)

# Splitting Data into Predictors and Response
predictors = dataset.iloc[:, 0:9]
response = dataset.iloc[:, 9]

# Check for Imbalance
dataset_eda = dataset.copy()
dataset_eda['classification'].value_counts()

# Feature Selection
model = RandomForestClassifier(random_state=2)
cv = StratifiedKFold(n_splits=5, shuffle=True)
GARF = GeneticSelectionCV(
            model,
            cv=cv,
            verbose=0,
            scoring="accuracy",
            max_features=9,
            n_population=100,
            crossover_proba=0.5,
            mutation_proba=0.2,
            n_generations=50,
            crossover_independent_proba=0.5,
            mutation_independent_proba=0.04,
            tournament_size=3,
            n_gen_no_change=10,
            caching=True,
```

```python
                n_jobs=-1)
GARF = GARF.fit(predictors, response)
print('Features:', predictors.columns[GARF.support_])

# Drop Non-Selected Features
predictors_ga = predictors.drop(['Interference Checked'], axis = 1)
print(predictors_ga)

# Data Splitting
seed = 2
test_size = 0.20
X_train, X_test, y_train, y_test = train_test_split(predictors, response,
                                                    test_size=test_size,
                                                    random_state=seed)

X_train_ga, X_test_ga, y_train_ga, y_test_ga = train_test_split(predictors_ga, response,
                                                                test_size=test_size,
                                                                random_state=seed)

# Check Number of Rows
X_train.shape[0], X_test.shape[0], X_train_ga.shape[0], X_test_ga.shape[0]

# Feature Scaling
scaler = StandardScaler()
X_train_std = scaler.fit_transform(X_train)
X_train_scaled = pd.DataFrame(X_train_std, index=X_train.index, columns=X_train.columns)
print(X_train_scaled)

# Feature Scaling for GA Model
X_train_gastd = scaler.fit_transform(X_train_ga)
X_train_gascaled = pd.DataFrame(X_train_gastd, index=X_train_ga.index,
                                columns=X_train_ga.columns)
print(X_train_gascaled)

# Class Balancing
sm = SMOTE(random_state=2)
X_train_sm, y_train_sm = sm.fit_resample(X_train, y_train)
X_train_sm_scaled, y_train_sm_scaled = sm.fit_resample(X_train_scaled, y_train)
X_train_all, y_train_all = sm.fit_resample(X_train_gascaled, y_train_ga)

# ---------------- Random Forest (No Feature Scaling) ------------- #
model = RandomForestClassifier(random_state=2)
warnings.filterwarnings('ignore')

# Hyperparameter Tuning
parameters = {
    'n_estimators': [50, 100, 150, 200],
    'max_features': ['sqrt', 'log2', None],
    'max_depth': [3, 6, 9],
    'max_leaf_nodes': [3, 6, 9],
}

# For Model 1
cv = StratifiedKFold(n_splits=5, shuffle=True)
model1 = GridSearchCV(model, parameters, cv=cv)
model1.fit(X_train, y_train)
print(model1.best_params_)
model1.best_estimator_.score(X_test, y_test)

# For Model 1 GA
model1_ga = GridSearchCV(model, parameters, cv=cv)
model1_ga.fit(X_train_ga, y_train_ga)
print(model1_ga.best_params_)
model1_ga.best_estimator_.score(X_test_ga, y_test_ga)

# Make Predictions for Test Data
y_pred = model1.predict(X_test)
y_pred_ga = model1_ga.predict(X_test_ga)

# Evaluate Predictions for Model 1
mcc = matthews_corrcoef(y_test_ga,y_pred_ga)
confusion_mat = confusion_matrix(y_test_ga,y_pred_ga)
print("MCC is:",mcc)
print("Confusion Matrix")
print(confusion_mat)
print(classification_report(y_test_ga,y_pred_ga))

# Evaluate Predictions for Model 1 GA
mcc = matthews_corrcoef(y_test_ga,y_pred_ga)
confusion_mat = confusion_matrix(y_test_ga,y_pred_ga)
print("MCC is:",mcc)
print("Confusion Matrix")
print(confusion_mat)
print(classification_report(y_test_ga,y_pred_ga))

# --- For Model 1 ---
fpr, tpr, threshold = roc_curve(y_test, y_pred, pos_label=1)
random = [0 for i in range(len(y_test))]
p_fpr, p_tpr, _ = roc_curve(y_test, random, pos_label=1)
auc_score = roc_auc_score(y_test, y_pred)*100

print("AUC Score: ", auc_score)
```

```

# Plot ROC Curves
plt.plot(fpr, tpr, linestyle='--',color='blue', label='Random Forest')
plt.plot(p_fpr, p_tpr, linestyle='--', color='black')

# Title and Labels
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')

#Legend
plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show()

# --- For Model 1 GA ---
fpr, tpr, threshold = roc_curve(y_test_ga, y_pred_ga, pos_label=1)
random = [0 for i in range(len(y_test_ga))]
p_fpr, p_tpr, _ = roc_curve(y_test_ga, random, pos_label=1)
auc_score = roc_auc_score(y_test_ga, y_pred_ga)*100
print("AUC Score: ", auc_score)

# Plot ROC Curves
plt.plot(fpr, tpr, linestyle='--',color='blue', label='Random Forest')
plt.plot(p_fpr, p_tpr, linestyle='--', color='black')

# Title and Labels
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')

#Legend
plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show()

# ---------------- Random Forest (With Feature Scaling) ------------- #
#For Model 2
model2 = GridSearchCV(model, parameters, cv=cv)
model2.fit(X_train_scaled, y_train)
print(model2.best_params_)
model2.best_estimator_.score(X_test, y_test)

# For Model 2 GA
model2_ga = GridSearchCV(model, parameters, cv=cv)
model2_ga.fit(X_train_gascaled, y_train_ga)
print(model2_ga.best_params_)
model2_ga.best_estimator_.score(X_test_ga, y_test_ga)

# Make Predictions for Test Data
y_pred = model2.predict(X_test)
y_pred_ga = model2_ga.predict(X_test_ga)

# Evaluate Predictions for Model 2
mcc = matthews_corrcoef(y_test_ga,y_pred_ga)
confusion_mat = confusion_matrix(y_test_ga,y_pred_ga)
print("MCC is:",mcc)
print("Confusion Matrix")
print(confusion_mat)
print(classification_report(y_test_ga,y_pred_ga))

# Evaluate Predictions for Model 2 GA
mcc = matthews_corrcoef(y_test_ga,y_pred_ga)
confusion_mat = confusion_matrix(y_test_ga,y_pred_ga)
print("MCC is:",mcc)
print("Confusion Matrix")
print(confusion_mat)
print(classification_report(y_test_ga,y_pred_ga))

# --- For Model 2 ---
fpr, tpr, threshold = roc_curve(y_test, y_pred, pos_label=1)
random = [0 for i in range(len(y_test))]
p_fpr, p_tpr, _ = roc_curve(y_test, random, pos_label=1)
auc_score = roc_auc_score(y_test, y_pred)*100

print("AUC Score: ", auc_score)

# Plot ROC Curves
plt.plot(fpr, tpr, linestyle='--',color='blue', label='Random Forest')
plt.plot(p_fpr, p_tpr, linestyle='--', color='black')

# Title and Labels
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')

#Legend
plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show()

# --- For Model 2 GA ---
```

```
fpr, tpr, threshold = roc_curve(y_test_ga, y_pred_ga, pos_label=1)
random = [0 for i in range(len(y_test_ga))]
p_fpr, p_tpr, _ = roc_curve(y_test_ga, random, pos_label=1)
auc_score = roc_auc_score(y_test_ga, y_pred_ga)*100
print("AUC Score: ", auc_score)

# Plot ROC Curves
plt.plot(fpr, tpr, linestyle='--',color='blue', label='Random Forest')
plt.plot(p_fpr, p_tpr, linestyle='--', color='black')

# Title and Labels
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')

#Legend
plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show()

# ------------------ Random Forest (With Smote) --------------- #
# For Model 3
model3 = GridSearchCV(model, parameters, cv=cv)
model3.fit(X_train_sm, y_train_sm)
print(model3.best_params_)
model3.best_estimator_.score(X_test, y_test)

# For Model 3 GA
model3_ga = GridSearchCV(model, parameters, cv=cv)
model3_ga.fit(X_train_ga, y_train_ga)
print(model3_ga.best_params_)
model3_ga.best_estimator_.score(X_test_ga, y_test_ga)

# Make Predictions for Test Data
y_pred = model3.predict(X_test)
y_pred_ga = model3_ga.predict(X_test_ga)

# Evaluate Predictions for Model 3
mcc = matthews_corrcoef(y_test, y_pred)
confusion_mat = confusion_matrix(y_test,y_pred)
print("MCC is:",mcc)
print("Confusion Matrix")
print(confusion_mat)
print(classification_report(y_test,y_pred))

# Evaluate Predictions for Model 3 GA
mcc = matthews_corrcoef(y_test_ga,y_pred_ga)
confusion_mat = confusion_matrix(y_test_ga,y_pred_ga)
print("MCC is:",mcc)
print("Confusion Matrix")
print(confusion_mat)
print(classification_report(y_test_ga,y_pred_ga))

# --- For Model 3 ---
fpr, tpr, threshold = roc_curve(y_test, y_pred, pos_label=1)
random = [0 for i in range(len(y_test))]
p_fpr, p_tpr, _ = roc_curve(y_test, random, pos_label=1)
auc_score = roc_auc_score(y_test, y_pred)*100

print("AUC Score: ", auc_score)

# Plot ROC Curves
plt.plot(fpr, tpr, linestyle='--',color='blue', label='Random Forest')
plt.plot(p_fpr, p_tpr, linestyle='--', color='black')

# Title and Labels
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')

#Legend
plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show()

# --- For Model 3 GA ---
fpr, tpr, threshold = roc_curve(y_test_ga, y_pred_ga, pos_label=1)
random = [0 for i in range(len(y_test_ga))]
p_fpr, p_tpr, _ = roc_curve(y_test_ga, random, pos_label=1)
auc_score = roc_auc_score(y_test_ga, y_pred_ga)*100
print("AUC Score: ", auc_score)

# Plot ROC Curves
plt.plot(fpr, tpr, linestyle='--',color='blue', label='Random Forest')
plt.plot(p_fpr, p_tpr, linestyle='--', color='black')

# Title and Labels
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')

#Legend
```

```
plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show()

# ---------------- Random Forest (All Applied) ------------- #
# For Model 4
model4 = GridSearchCV(model, parameters, cv=cv)
model4.fit(X_train_sm_scaled, y_train_sm_scaled)
print(model4.best_params_)
model4.best_estimator_.score(X_test, y_test)

# For Model 4 GA
model4_ga = GridSearchCV(model, parameters, cv=cv)
model4_ga.fit(X_train_all, y_train_all)
print(model4_ga.best_params_)
model4_ga.best_estimator_.score(X_test_ga, y_test_ga)

# Make Predictions for Test Data
y_pred = model4.predict(X_test)
y_pred_ga = model4_ga.predict(X_test_ga)

# Evaluate Predictions for Model 4
mcc = matthews_corrcoef(y_test, y_pred)
confusion_mat = confusion_matrix(y_test,y_pred)
print("MCC is:",mcc)
print("Confusion Matrix")
print(confusion_mat)
print(classification_report(y_test,y_pred))

# Evaluate Predictions for Model 4 GA
mcc = matthews_corrcoef(y_test_ga,y_pred_ga)
confusion_mat = confusion_matrix(y_test_ga,y_pred_ga)
print("MCC is:",mcc)
print("Confusion Matrix")
print(confusion_mat)
print(classification_report(y_test_ga,y_pred_ga))

# --- For Model 4 ---
fpr, tpr, threshold = roc_curve(y_test, y_pred, pos_label=1)
random = [0 for i in range(len(y_test))]
p_fpr, p_tpr, _ = roc_curve(y_test, random, pos_label=1)
auc_score = roc_auc_score(y_test, y_pred)*100
print("AUC Score: ", auc_score)

# Plot ROC Curves
plt.plot(fpr, tpr, linestyle='--',color='blue', label='Random Forest')
plt.plot(p_fpr, p_tpr, linestyle='--', color='black')

# Title and Labels
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')

#Legend
plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show()

# --- For Model 4 GA ---
fpr, tpr, threshold = roc_curve(y_test_ga, y_pred_ga, pos_label=1)
random = [0 for i in range(len(y_test_ga))]
p_fpr, p_tpr, _ = roc_curve(y_test_ga, random, pos_label=1)
auc_score = roc_auc_score(y_test_ga, y_pred_ga)*100
print("AUC Score: ", auc_score)

# Plot ROC Curves
plt.plot(fpr, tpr, linestyle='--',color='blue', label='Random Forest')
plt.plot(p_fpr, p_tpr, linestyle='--', color='black')

# Title and Labels
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')

#Legend
plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show()

# ---------------- Saving the Pipeline into a File ------------- #
pickle.dump(model3_ga, open('garf.pkl', 'wb'))
```

4. **HTML: index.html**

```
{% load static %}

<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <title>Toxicheck - Nanotoxicity Classification System</title>
    <link rel="icon" type="images/x-icon"
          href="https://cdn-icons-png.flaticon.com/512/4689/4689000.png" />
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
          integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
          crossorigin="anonymous" rel="stylesheet">
    <link rel="stylesheet"
          href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">
    <link rel="stylesheet" href="{% static 'main.css' %}">
    <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.min.js"
            integrity="sha384-IQsoLXl5PILFhosVNubq5LC7Qb9DXgDA9i+tQ8Zj3iwWAwPtgFTxbJ8NT4GN1R8p"
            crossorigin="anonymous"></script>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.min.js"
            integrity="sha384-cVKIPhGWiC2Al4u+LWgxfKTRIcfu0JTxR+EQDz/bgldoEyl4H0zUF0QKbrJ0EcQF"
            crossorigin="anonymous"></script>
  </head>

  <body>
    <nav class="navbar navbar-dark" style="background: #2365C2;">
      <div class="container-fluid">
        <a class="navbar-brand" href="{% url 'home'%}">
        <img src="{% static 'logo.png' %}" alt="" width="30" height="30"
        class="d-inline-block align-text-top" style="margin-right: 10px;">
        Toxicheck - Nanotoxicity Classification System </a>
      </div>
    </nav>

    <div class="container">
      <div class="row">
        <div class="col-md-5">
          <img src="static/logo.png" style="width: 370px; height: 370px; display: block;
          margin-left: auto; margin-right: auto; margin-top: 60px;" alt="Toxicheck Icon">
        </div>

        <div class="col-md-7">
          <div class="container-fluid" style="margin-top: 120px; margin-bottom: 120px;">
            <h5 style="color: #2365C2; ">Machine Learning</b></h5>
            <h1 style="margin-bottom: 15px;">Nanotoxicity Classification System
            <b>Toxicheck     </b></h1>
            <p>Due to the innate volatility of nanomaterials (NM), toxicity testing is an
              important step in engineering NMs to ensure that they are safe to organisms
              and the environment.</p>
            <a href="{% url 'predict' %}" class="btn button" style="background: #2365C2;
             color: white; margin-top: 10px;">Test Now</a>
          </div>
        </div>
      </div>
    </div>
  </body>

  <footer class="text-center text-lg-start text-dark">
    <!-- Section: Links -->
    <section class="d-flex justify-content-between p-3 text-white"
    style="background-color: #2365C2">
      <!-- Left -->
      <div class="me-5">
        <span>If you have questions and/or feedback, reach us through: </span>
      </div>

      <!-- Right -->
      <div>
        <a href="https://tinyurl.com/5pc0d35" class="text-white me-4">
        <i class="fa fa-google" aria-hidden="true"></i></a>
      </div>
    </section>

    <!-- Section: Contacts -->
    <section class="">
      <div class="container text-center text-md-start mt-4">
        <div class="row md-3">
          <div class="col-md-5 col-lg-5 col-xl-5 mx-auto mb-4">
            <h6 class="text-uppercase fw-bold">System Project</h6>
              <p>
                Toxicheck simulates the effect of a nanomaterial as described by its
                physicochemical properties to the cell viability of a cell-based assay.
              </p>
          </div>

          <div class="col-md-5 col-lg-5 col-xl-5 mx-auto mb-md-0 mb-4">
            <h6 class="text-uppercase fw-bold">Contact Information</h6>
              <p><i class="fa fa-envelope"></i>
                jebarcellano@up.edu.ph - John Derick Barcellano</p>
              <p style="margin-top:-15px;">
                <i class="fa fa-phone"></i> +63 977 294 5883</p>
          </div>
        </div>
      </div>
    </section>
  </footer>
</html>
```

5. **HTML: homepage.html**

```
{% load static %}

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <title>Toxicheck - Nanotoxicity Classification System</title>
    <link rel="icon" type="images/x-icon"
          href="https://cdn-icons-png.flaticon.com/512/4689/4689000.png" />
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
          integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
          crossorigin="anonymous" rel="stylesheet">
    <link rel="stylesheet"
      href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">
    <link rel="stylesheet" href="{% static 'main.css' %}">
    <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.min.js"
            integrity="sha384-IQsoLXl5PILFhosVNubq5LC7Qb9DXgDA9i+tQ8Zj3iwWAwPtgFTxbJ8NT4GN1R8p"
            crossorigin="anonymous"></script>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.min.js"
            integrity="sha384-cVKIPhGWiC2Al4u+LWgxfKTRIcfu0JTxR+EQDz/bgldoEyl4H0zUF0QKbrJ0EcQF"
            crossorigin="anonymous"></script>
  </head>

  <body>
    <nav class="navbar navbar-dark" style="background: #2365C2;">
      <div class="container-fluid">
        <a class="navbar-brand" href="{% url 'home'%}">
          <img src="{% static 'logo.png' %}" alt="" width="30" height="30"
          class="d-inline-block align-text-top" style="margin-right: 10px;">
          Toxicheck - Nanotoxicity Classification System </a>
      </div>
    </nav>

    <div class="container-fluid">
      <div class="row">
        <!-- Left Panel -->
        <div class="card col-md-6">
          <h2 style="margin-bottom: 12px">Input Form</h2>
          <form action="predict" method="post" id = "form">
            {% csrf_token %}

            <div class="row pb-3">
              <div class="col-md-4">
                <label title="NP Type specifies whether the nanomaterial is Organic (0)
                or Inorganic (1).">
                  NP Type    :</label>
                <input type="number" class="form-control" name="NP Type"
                placeholder="Enter 0 or 1" value="{{nptype}}" min = "0" max="1" required>
              </div>

              <div class="col-md-4">
                <label title="Diameter specifies the size of the nanomaterial in nanometers.">
                  Diameter (nm)    :</label>
                <input type="number" class="form-control" name="Diameter"
                placeholder="Enter 1 to 957" value="{{diameter}}" min = "1" step="any" required>
              </div>

              <div class="col-md-4">
                <label title="Concentration specifies the amount of the nanomaterial
                being applied to the cell-based assay. It is measured in micrometers.">
                  Concentration ( m )    :</label>
                <input type="number" class="form-control" name="Concentration"
                placeholder="Enter 0 to 15000" value="{{concentration}}" min = "0"
                step="any" required>
              </div>
            </div>

            <div class="row pb-3">
              <div class="col-md-4">
                <label title="Colloidal stability specifies whether the nanomaterial
                is colloidally stable (1) or not (0). Being stable allows increased
                diffusive capability in the brain microenvironment.">
                  Colloidal Stability    :</label>
                <input type="number" class="form-control" name="Colloidal Stability Checked"
                placeholder="Enter 0 or 1" value="{{colloidal}}" min = "0" max="1" required>
              </div>

              <div class="col-md-4">
                <label title="Surface charge specifies whether the nanomaterial has a
                 negative (0) or a positive (1) charge. This property determines cellular
                 uptake, biodistribution, and interaction with other biological environments.">
                  Surface Charge    :</label>
                <input type="number" class="form-control" name="Positive Control"
                placeholder="Enter 0 or 1" value="{{positive}}" min = "0" max="1" required>
              </div>

              <div class="col-md-4">
```

```html
                  <label title="Cell culture specifies whether the cell used in the assay
                   is a primary cell (0) or a cell line (1). Primary cells are isolated from
                    parental tissues, while cell lines are cultures from primary cells.">
                    Cell Culture    :</label>
                  <input type="number" class="form-control" name="Cell Culture"
                  placeholder="Enter 0 or 1" value="{{culture}}" min = "0" max="1" required>
                </div>
              </div>

              <div class="row pb-3">
                <div class="col-md-4">
                  <label title="Cell type specifies whether the cell used in the assay
                  is a human cell (0) or an animal cell (1).">
                    Cell Type    :</label>
                  <input type="number" class="form-control" name="Cell Type"
                  placeholder="Enter 0 or 1"value="{{celltype}}" min = "0" max="1" required>
                </div>

                <div class="col-md-4">
                  <label title="Cell age specifies whether the cell used in the assay is
                   adult (0) or still embryonic (1).">
                    Cell Age    :</label>
                  <input type="number" class="form-control" name="Cell Age"
                  placeholder="Enter 0 or 1" value="{{age}}" min = "0" max="1" required>
                </div>

                <div class="col-md-4">
                  <label title="Exposure time specifies the amount of time the nanoparticle
                   is exposed to the cell-based assay in hours.">
                    Exposure Time (Hrs)    :</label>
                  <input type="number" class="form-control" name="Exposure Time"
                  placeholder="Enter 1 to 336" value="{{exposuretime}}" min = "1" required>
                </div>
              </div>

              <span title="This field prints non-toxic or toxic depending on the model's
               prediction on the assay's cell viability after being exposed to the nanomaterial.">
                Prediction    : <b> {{ans}} </b></span>

              <br><br>
              <button type="submit" class="btn button" style="background: #2365C2;
              color: white;">Submit</button>
              <a href="{% url 'predict' %}" class="btn button" style="background: #2365C2;
              color: white; margin-left: 8px;">Reset</a>
            </form>
          </div>

          <!-- Right Panel -->
          <div class="card col-md-6">
            <h2 style="margin-bottom: 12px">Model Details</h2>
            <span style="margin-bottom: 12px;"> This section shows the performance metrics
              of GA-RF with SMOTE. </span>
            <div id="output">
              <div class="row pb-2">
                <div class="col-md-4">
                  Accuracy: <b>{{accuracy}}%</b>
                </div>

                <div class="col-md-4">
                  Precision: <b>{{precision}}%</b>
                </div>

                <div class="col-md-4">
                  Recall: <b>{{recall}}%</b>
                </div>
              </div>

              <div class="row pb-2">
                <div class="col-md-4">
                  F1 Score: <b>{{f1}}%</b>
                </div>

                <div class="col-md-8">
                  MCC: <b>{{mcc}}</b>
                </div>
              </div>

              <img src="{% static 'roc_curve.png' %}" alt="ROC" width="300" height="225">
              <img src="{% static 'confusion_matrix.png' %}" alt="CM" width="300" height="225">
            </div>
          </div>
        </div>
      </div>
    </body>

    <footer class="text-center text-lg-start text-dark">
      <!-- Section: Links -->
      <section class="d-flex justify-content-between p-3 text-white"
      style="background-color: #2365C2">
        <!-- Left -->
        <div class="me-5">
          <span>If you have questions and/or feedback, reach us through: </span>
        </div>

        <!-- Right -->
        <div>
          <a href="https://tinyurl.com/5pc0d35" class="text-white me-4">
          <i class="fa fa-google" aria-hidden="true"></i></a>
        </div>
      </section>

      <!-- Section: Contacts -->
      <section class="">
        <div class="container text-center text-md-start mt-4">
          <div class="row md-3">
            <div class="col-md-5 col-lg-5 col-xl-5 mx-auto mb-4">
              <h6 class="text-uppercase fw-bold">System Project</h6>
                <p>
                  Toxicheck simulates the effect of a nanomaterial as described by its
                  physicochemical properties to the cell viability of a cell-based assay.
                </p>
            </div>

            <div class="col-md-5 col-lg-5 col-xl-5 mx-auto mb-md-0 mb-4">
              <h6 class="text-uppercase fw-bold">Contact Information</h6>
                <p><i class="fa fa-envelope"></i>
                  jebarcellano@up.edu.ph - John Derick Barcellano</p>
                <p style="margin-top:-15px;">
                  <i class="fa fa-phone"></i> +63 977 294 5883</p>
            </div>
          </div>
        </div>
      </section>
    </footer>
  </html>
```

6. **CSS: main.css**

```
body{
    color: black;
    background-color: white;
}

footer{
    background-color: #ECEFF1;
}

.card{
    background: #fff;
    width: 45%;
    border-radius: 20px;
    padding: 25px;
    margin: 25px;
    text-align: left;
    box-shadow: 0px 0px 10px rgba(0,0,0,0.3);
    transition: all 300ms ease;
}

.card:hover{
    box-shadow: none;
}

.fa{
    margin-right: 5px;
}

a{
    text-decoration: none;
}

label{
    margin-bottom: 8px;
}
```

7. **Django: apps.py**

```
from django.apps import AppConfig

class NcsConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'toxicheck'
```

8. **Django: urls.py**

```
from django.contrib import admin
from django.urls import path, include
from toxicheck import views

```

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.index, name = 'home'),
    path('predict', views.predict, name = 'predict'),
]
```

9. **Django: views.py**

```
from django.shortcuts import render

# ——— Importing Libraries ———
import math
import joblib
import pickle
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
from genetic_selection import GeneticSelectionCV
from imblearn.over_sampling import SMOTE
from scipy import stats
from sklearn.metrics import accuracy_score, precision_score, f1_score, recall_score
from sklearn.metrics import confusion_matrix, roc_auc_score, roc_curve
from sklearn.metrics import matthews_corrcoef
from sklearn.model_selection import GridSearchCV, train_test_split, StratifiedKFold
from sklearn.preprocessing import LabelEncoder, MultiLabelBinarizer, StandardScaler
from sklearn.ensemble import RandomForestClassifier
matplotlib.use('Agg')

# ——— Importing the Model ———
model = joblib.load('garf-1.pkl')

def predict(request):
    # ——— Calculating Metrics ———
    dataset = pd.read_csv('np.csv')

    # Splitting Data into Predictors and Response
    predictors = dataset.iloc[:, 0:9]
    response = dataset.iloc[:, 9]

    # Train and Test Split
    seed = 2
    test_size = 0.20
    X_train, X_test, y_train, y_test = train_test_split(predictors, response,
                                                         test_size=test_size,
                                                         random_state=seed)

    # Make Predictions
    y_pred = model.predict(X_test)

    # Calculating Metrics
    accuracy = ("%.2f"% (accuracy_score(y_test,y_pred)*100))
    precision = ("%.2f"% (precision_score(y_test,y_pred,average='weighted')*100))
    f1 = ("%.2f"% (f1_score(y_test,y_pred,average='weighted')*100))
    recall = ("%.2f"% (recall_score(y_test,y_pred,average='weighted')*100))
    mcc = ("%.4f"% matthews_corrcoef(y_test, y_pred))
    fpr, tpr, thresholds = roc_curve(y_test,y_pred)
    auc = roc_auc_score(y_test, y_pred)

    # Plot the ROC curve
    fig, ax = plt.subplots()
    ax.plot(fpr, tpr, label='ROC Curve (area = %.2f)' % auc)
    ax.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random guess')
    ax.set_title('ROC Curve')
    ax.set_xlabel('False Positive Rate')
    ax.set_ylabel('True Positive Rate')
    ax.grid(True)
    ax.legend()
    plt.savefig('static/roc_curve.png')
    plt.close(fig)

    # Generate confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    fig, ax = plt.subplots()
    sns.heatmap(cm, annot=True, fmt='g', cmap='Blues', ax=ax)
    ax.set_xlabel('Predicted')
    ax.set_ylabel('Actual')
    plt.savefig('static/confusion_matrix.png')
    plt.close(fig)

    context = {
        'accuracy':accuracy,
        'precision':precision,
        'recall':recall,
        'f1': f1,
        'auc':auc,
        'mcc':mcc,
    }

    if request.method=='POST':
        temp={}
        temp['NP Type']=float(request.POST.get('NP Type'))
        temp['Diameter']=float(request.POST.get('Diameter'))
        temp['Concentration']=float(request.POST.get('Concentration'))
        temp['Cell Culture']=float(request.POST.get('Cell Culture'))
        temp['Cell Type']=float(request.POST.get('Cell Type'))
        temp['Cell Age']=float(request.POST.get('Cell Age'))
        temp['Exposure Time']=float(request.POST.get('Exposure Time'))
        temp['Colloidal Stability Checked']=float(request.POST.get('Colloidal Stability Checked'))
        temp['Positive Control']=float(request.POST.get('Positive Control'))
        testdata=pd.DataFrame({'x':temp}).transpose()
        sorteddata=testdata [['NP Type','Diameter','Concentration','Cell Culture',
                              'Cell Type','Cell Age','Exposure Time',
                              'Colloidal Stability Checked','Positive Control']]

        score = model.predict(sorteddata)[0]
        if score == 0:
            ans = "Non-Toxic"
        else:
            ans = "Toxic"

        # ---- Calculating Metrics ----
        dataset = pd.read_csv('np.csv')

        # Splitting Data into Predictors and Response
        predictors = dataset.iloc[:, 0:9]
        response = dataset.iloc[:, 9]

        # Train and Test Split
        seed = 2
        test_size = 0.20
        X_train, X_test, y_train, y_test = train_test_split(predictors, response,
                                                            test_size=test_size,
                                                            random_state=seed)

        # Make Predictions
        y_pred = model.predict(X_test)

        # Calculating Metrics
        accuracy = ("%.2f"% (accuracy_score(y_test,y_pred)*100))
        precision = ("%.2f"% (precision_score(y_test,y_pred,average='weighted')*100))
        f1 = ("%.2f"% (f1_score(y_test,y_pred,average='weighted')*100))
        recall = ("%.2f"% (recall_score(y_test,y_pred,average='weighted')*100))
        mcc = ("%.4f"% matthews_corrcoef(y_test, y_pred))
        fpr, tpr, thresholds = roc_curve(y_test,y_pred)
        auc = roc_auc_score(y_test, y_pred)

        # Plot the ROC curve
        fig, ax = plt.subplots()
        ax.plot(fpr, tpr, label='ROC Curve (area = %.2f)' % auc)
        ax.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random guess')
        ax.set_title('ROC Curve')
        ax.set_xlabel('False Positive Rate')
        ax.set_ylabel('True Positive Rate')
        ax.grid(True)
        ax.legend()
        plt.savefig('static/roc_curve.png')
        plt.close(fig)

        # Generate confusion matrix
        cm = confusion_matrix(y_test, y_pred)
        fig, ax = plt.subplots()
        sns.heatmap(cm, annot=True, fmt='g', cmap='Blues', ax=ax)
        ax.set_xlabel('Predicted')
        ax.set_ylabel('Actual')
        plt.savefig('static/confusion_matrix.png')
        plt.close(fig)

        context = {
            'ans':ans,
            'accuracy':accuracy,
            'precision':precision,
            'recall':recall,
            'f1': f1,
            'auc':auc,
            'mcc':mcc,
            'nptype': temp['NP Type'],
            'diameter': temp['Diameter'],
            'concentration': temp['Concentration'],
            'culture': temp['Cell Culture'],
            'celltype': temp['Cell Type'],
            'age': temp['Cell Age'],
            'exposuretime': temp['Exposure Time'],
            'colloidal': temp['Colloidal Stability Checked'],
            'positive': temp['Positive Control'],
        }
        return render(request, 'toxicheck/homepage.html', context)
    return render(request, 'toxicheck/homepage.html', context)

def index(request):
    return render(request, 'toxicheck/index.html',)
```

XI. Acknowledgment

Words cannot express my gratitude to Lady Edronalee, Cornelius, Ivan, and Heidi. They were the people who helped me immensely during the development of this project and gave me the hope that I needed to finish it. Thank you for teaching and guiding me throughout this process. I sincerely cannot thank you enough.

I extend my sincerest appreciation to my thesis adviser, Ms. Perlita Gasmen, for her guidance, encouragement, and patience whenever I make mistakes. Thank you for supporting me and pushing me to do my best.

To Sir John Bagnol and his advisee, Kryzze Lee, who provided the inspiration and resources for my topic. Thank you for helping me understand the key concepts of this study and giving me your support as we collaborate on our projects.

To my family and my friends on campus, who constantly reminded me that I could do this. I would like to express my gratitude to all of you for supporting me in different ways and for pushing me to never give up when I was in my lowest moments.

To my other friends and my best friend, Dizon, you were also a big part of my inspiration to finish this project. You might not be aware of this, but I valued every minute that you gave me when I wanted to unwind.

And to God, for whom I am deeply thankful, for blessing me with all these people who are there to support me. Even though I am not the most religious person, He still did not abandon me throughout this journey.

Finally, I would like to acknowledge all individuals who have played a part, no matter how big or small, in inspiring me during my college life. This endeavor would not have been possible without all of your contributions and support. And for that, I am truly grateful to all of you.